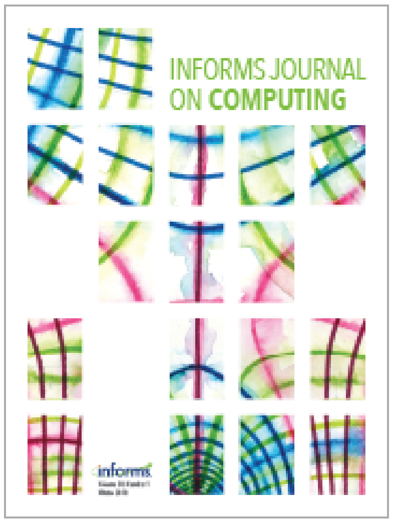


INFORMS Journal on Computing

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

Tyler Perini, Natashia Boland, Diego Pecin, Martin Savelsbergh

informs
http://pubsonline.informs.org/journal/ijoc/

INFORMS JOURNAL ON COMPUTING
Articles in Advance, pp. 1–24
ISSN 1091-9856 (print), ISSN 1526-5528 (online)

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

Tyler Perini,[a] Natashia Boland,[a] Diego Pecin,[a] Martin Savelsbergh[a]

[a] H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332

Contact: perinita@gatech.edu, http://orcid.org/0000-0002-8502-8111 (TP); natashia.boland@gmail.com, http://orcid.org/0000-0002-6867-6125 (NB); diego.pecin@gmail.com, http://orcid.org/0000-0001-7965-7264 (DP); martin.savelsbergh@isye.gatech.edu, http://orcid.org/0000-0001-7031-5516 (MS)

Received: December 12, 2017
Revised: October 16, 2018; December 20, 2018
Accepted: December 27, 2018
Published Online in Articles in Advance: August 7, 2019

https://doi.org/10.1287/ijoc.2019.0887

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.

History: Accepted by David Woodruff, Area Editor for Software Tools.
Funding: Financial support was received from the National Science Foundation Graduate Research Fellowship [Grant DGE-1650044].
Supplemental Material: The online supplement is available at https://doi.org/10.1287/ijoc.2019.0887.

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), healthcare (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), and Zhou et al. (2011). This interest largely concerns continuous multiobjective optimization problems. The number of evolutionary algorithms dealing with discrete multiobjective optimization problems, which will be our focus, is considerably smaller.

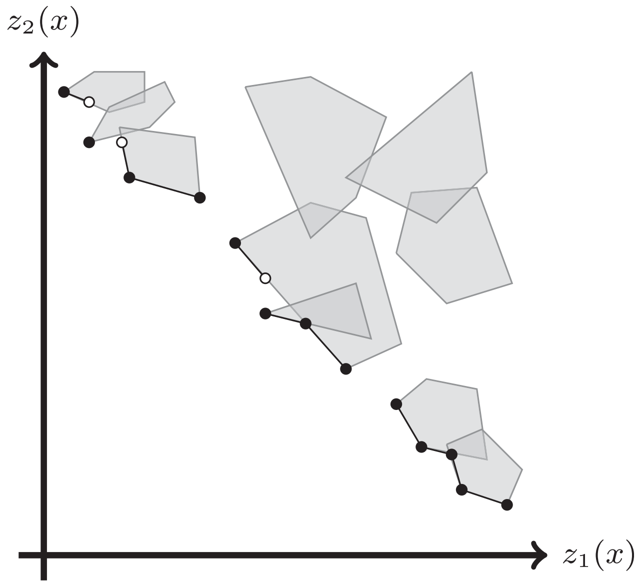
We focus on problems with linear objective functions and constraints and on exact algorithms, which are guaranteed, in theory, to produce the complete NDF. Biobjective integer linear programming has been studied for several decades now; see, for example, the surveys of Ehrgott and Gandibleux (2000, 2002). 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 30 years, at least a dozen methods for generating the NDF for pure integer problems have been developed.

Biobjective mixed integer linear programs (BOMIPs), by contrast, have only recently received vigorous interest, in part owing to their additional numerical challenges. BOMIP frontiers have a complex structure. 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, 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.

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*,

Figure 1. The Nondominated Frontier of a Biobjective Mixed Integer Linear Programs (BOMIP) Where Nondominated Line Segments Are Darkened



have emerged. Such methods have the advantage of being able to exploit advances in single-objective solver software because 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 (Cohon 1978, 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. The 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 biobjective linear program (BOLP) defined by fixing the integer part of the solution. The TSA thus requires a postprocessing procedure to provide a parsimonious description of the NDF.

The second algorithm published is the ε tabu-constraint method (εTCM) 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:

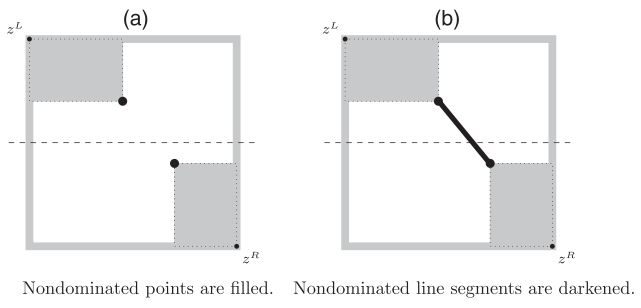
1. We propose a new criterion space search method for solving the BOMIP: the boxed line method (BLM). The method is designed to generalize the balanced box method (BBM; Boland et al. 2015a), which is a computationally effective method for pure integer biobjective integer program (BOIP). The BLM defaults to BBM in the absence of continuous variables. The key step of the BBM is illustrated in Figure 2(a): the rectangular search region (box) is split, and each half is searched for an NDP, which reduces the remaining search region to two boxes having a total area of 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 BLM, we seek to extend this NDP to the line segment in the NDF that contains it. Using the endpoints of this line segment gives a remaining search region consisting of two boxes, with a combined area of less than half that of the original box. Figure 2(b) 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 an enhancement that takes advantage of a property of the NDF encountered in many BOMIP instances, which can provide a significant improvement in algorithm runtime.

4. The benchmark instances originally proposed by Mavrotas and Diakoulaki (1998), which have been

Figure 2. The Key Step in the Balanced Box Method and the Boxed Line Method, Respectively


(a)
(b)
z^L
z^R
z^L
z^R
Nondominated points are filled.
Nondominated line segments are darkened.

Note. The remaining search regions (boxes) after the step are shaded.

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 if one of the algorithms is designed to exploit this structure. 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 BLM 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).

This paper is structured as follows. In Section 2, we introduce notation and key definitions. The basic variant of the BLM 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 in Section 4 along with its better upper bound for the number of IPs in Section 4.4. The enhancement designed for a structure common to many BOMIP instances is discussed in Section 5. In Section 6, we discuss numerical issues in implementation and how they relate to the structure of existing benchmark instances. 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 BOMIP as

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

where $z_1(x), z_2(x)$ are linear in x, and the feasible region is given by $\mathscr{X} \subseteq \mathbb{Z}^{n_I} \times \mathbb{R}^{n_C}$. We assume that $\mathscr{X}$ is nonempty and bounded. Note that a vector $x \in \mathscr{X}$ has n_I integer components and n_C continuous components, and when it is convenient to differentiate between the components, we use the convention $x = (x_I, x_C)$, where $x_I \in \mathbb{Z}^{n_I}$ and $x_C \in \mathbb{R}^{n_C}$. Let the projections of $\mathscr{X}$ onto the set of integer and real vectors be defined as $\mathscr{X}_I := \{x_I \in \mathbb{Z}^{n_I} : (x_I, x_C) \in \mathscr{X}$ for some $x_C \in \mathbb{R}^{n_C}\}$ and $\mathscr{X}_C := \{x_C \in \mathbb{R}^{n_C} : (x_I, x_C) \in \mathscr{X}$ for some $x_I \in \mathbb{Z}^{n_I}\}$, respectively. The feasible region $\mathscr{X}$ lies in the decision space $\mathbb{Z}^{n_I} \times \mathbb{R}^{n_C}$ or simply $\mathbb{R}^{n_I+n_C}$. By contrast, the image of $\mathscr{X}$ under $z(x)$, that is, $\mathscr{Y} := \{y \in \mathbb{R}^2 : y = z(x)$ for some $x \in \mathscr{X}\}$, lies in the criteria space $\mathbb{R}^2$.

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

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

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

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

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

are solved. Then $\eta = (\eta_1, \eta_2)$ is an NDP of (1). We note that in practice the second IP tends to solve very quickly. Lexicographic optimization in the order $z_2(x)$ and then $z_1(x)$ is similar. For positive vector $\lambda \in \mathbb{R}^2_+$, we refer to

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

as the *scalarized IP* with respect to λ. Let x^λ be an optimal solution to (3). Then $z(x^\lambda)$ is an NDP of (1). Not every NDP in the NDF can be found by such an IP. If, for a given NDP $z(x_N)$, 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 NDF can be described by nondominated line segments (NLSs), vertical gaps, and horizontal gaps. Define $L(z^1, z^2)$ to be the line segment connecting endpoints $z^1, z^2 \in \mathbb{R}^2$, that is, $L(z^1, z^2) := \{\xi z^1 + (1-\xi)z^2 : 0 \le \xi \le 1\}$, where the endpoints are ordered such that $z^1_1 \le z^2_1$. As defined, $L(z^1, z^2)$ is a closed line segment. When $z^1 \ne z^2$, we can also have an open line segment (neither z^1 nor z^2 is part of the line segment), and a half-open line segment (either z^1 or z^2 is not part of the line segment). We refer to z^1 and z^2 as *closed* if they belong to the line segment and as *open* otherwise. A line segment $L(z^1, z^2)$ is a point where $z^1 = z^2$.

A gap may appear in the NDF between two NDPs in the vertical or horizontal direction or both. Between an NDP and an open endpoint of a line segment in the NDF 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 NDP 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 $\{p^0, p^1, \ldots\}$ with $\lim_{n\to\infty} p_2^n = y^+$ (or both). A *horizontal gap* can be defined similarly.

We define S to be the index set of all feasible integer solutions in $\mathscr{X}_I$. Given $x_I \in X_I$, the BOLP obtained from fixing the integer variables to x_I is called the *slice problem* for x_I (Belotti et al. 2013). The NDF of a slice problem consists of a (connected) set of (closed) line segments; we call this a *slice*. The NDF of a slice problem for x_I is called the *slice for* x_I.[1] For a slice problem with index $s \in S$, we denote its slice by N^s. If a slice problem with index $s \in S$ contributes to the NDF of the BOMIP so that $N^s \cap \mathcal{N}$ is nonempty, then we write $N^s \cap \mathcal{N} = \{L_1^s, L_2^s, \ldots, L_{n(s)}^s\}$, where $n(s)$ is the number of line segments contributed to the NDF by the slice problem. (If $N^s \cap \mathcal{N}$ is empty, then $n(s) = 0$.) Each of the line segments L_i^s for some $s \in S$ and $i = 1, \ldots, n(s)$ is an NLS in the NDF. Note that a single maximal line segment in N^s may contribute several NLSs to the NDF, consisting of nonoverlapping sections of the line segment. We make the natural assumption that the set $N^s \cap \mathcal{N}$ consists of maximal line segments, where maximal refers to set inclusion, so $N^s \cap \mathcal{N}$ provides the minimum cardinality set of line segments that describes the slice's intersection with the NDF of the BOMIP. Given a BOMIP in the form of (1), we define the total number of NLSs in the NDF as

$$n^* := \sum_{s\in S} n(s). \tag{4}$$

3. Basic Method

Here we present the fundamental principles of the BLM. 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 online supplement 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 NDF exactly. In Section 6, we define an approximation—an ϵNDF—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 $z^L = \text{lexmin}\{(z_1(x), z_2(x)) : x \in \mathscr{X}\}$ and the lower-rightmost NDP $z^R = \text{lexmin}\{(z_2(x), z_1(x)) : x \in \mathscr{X}\}$. We refer to a rectangular region in criterion space as a *box* and describe it by its upper-leftmost and lower-rightmost corner points. The entire NDF 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 a box with $z_i^L = z_i^R$ for $i = 1$ or 2 (or both) *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 NDF. Otherwise, the two points are added to the current subset of the NDF, which we call $\tilde{\mathcal{N}}$, 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 an 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 BLM, we first describe the process for the BBM (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 $\tilde{\mathcal{N}}$ with any found NDPs and $\mathcal{Q}$ with remaining boxes for future processing. Suppose that 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 necessarily the NDPs found by initialization. The outer loop begins processing by choosing an arbitrary horizontal split line $z_2 = \mu$, where $\mu \in (z_2^R, z_2^L)$. We solve a lexicographic IP for an NDP on or below the split line, namely

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

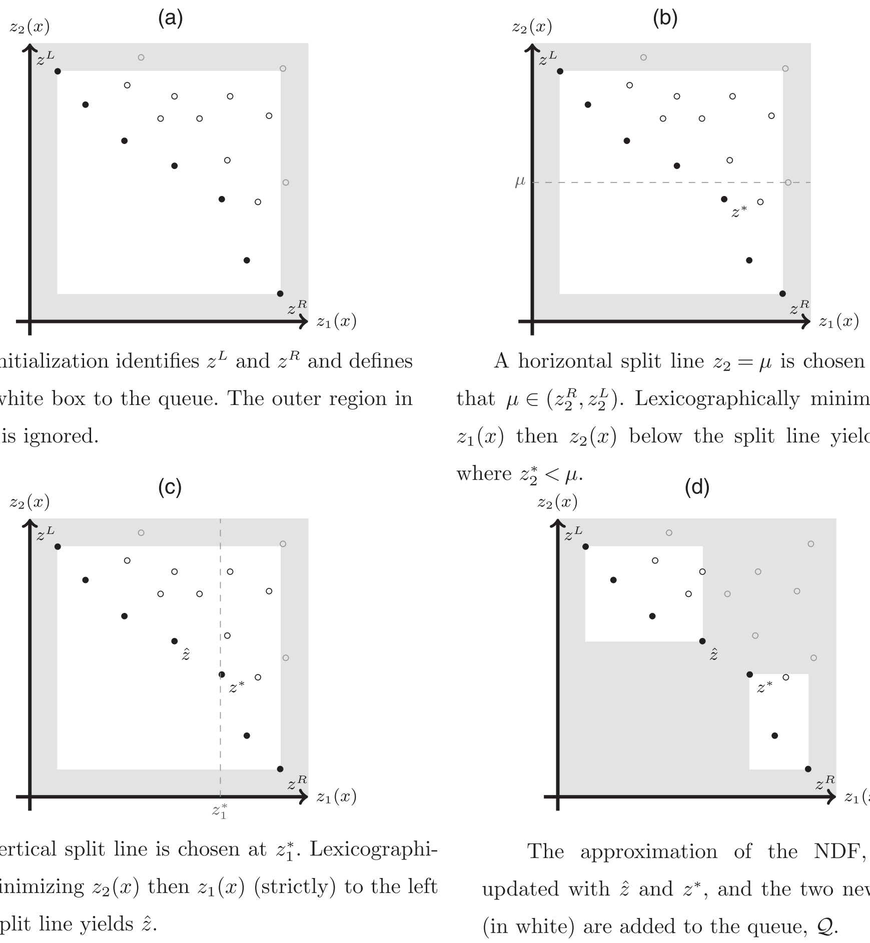
We note that the solution $x^R \in \mathscr{X}$, which maps to $z^R = z(x^R)$, is feasible for (5), so in practice we provide x^R as an initial feasible solution to the solver. The next step is to form a second split line, this time vertically at z_1^*, and solve a second lexicographic minimization[2] for the next NDP

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

The solution $x^L \in \mathscr{X}$ 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 $\tilde{\mathcal{N}}$ and $\mathcal{Q}$ accordingly. First, z^* and $\hat{z}$ are added to $\tilde{\mathcal{N}}$. 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 $z_2^* < \mu$.

Figure 3. Outer Loop Procedure for the Balanced Box Method and the Boxed Line Method When $z_2^* < \mu$



(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 $z_2 = \mu$ is chosen such that $\mu \in (z_2^R, z_2^L)$. 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 approximation of the NDF, $\tilde{\mathcal{N}}$, is updated with $\hat{z}$ and z^*, and the two new boxes (in white) are added to the queue, $\mathcal{Q}$.

The BLM follows this procedure when the first lexicographic minimization (5) returns NDP z^* such that $z_2^* < \mu$. Note that when there are discrete variables, it is possible for an arbitrary horizontal split line $z_2 = \mu$ 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 NDF. 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 NDF. 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^*, that is, 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 BLM.

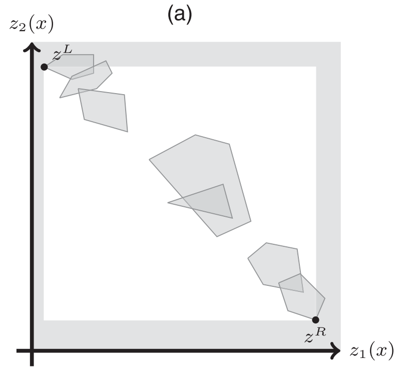
The outer loop calls the inner loop, which returns $L(z^1, z^2)$ (details on how this is done are given in Section 3.3). The inner loop returns endpoints z^1 and z^2, as well as the NDP that dominates each open endpoint, if any. The outer loop updates the current approximation of the NDF by adding $L(z^1, z^2)$ to $\tilde{\mathcal{N}}$ and updates the queue by adding up to two nontrivial boxes. Because 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 online supplement as Algorithm 1.

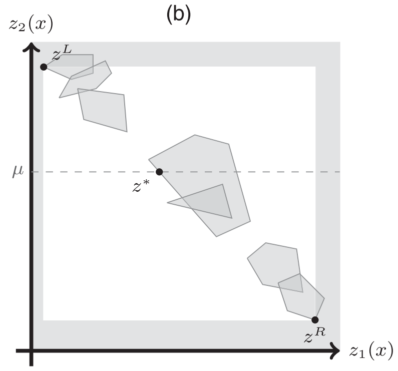
3.3. Inner Loop

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^*, that is, 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 5, and the pseudocode is presented in the online supplement as Algorithm 2. Here we provide a description.

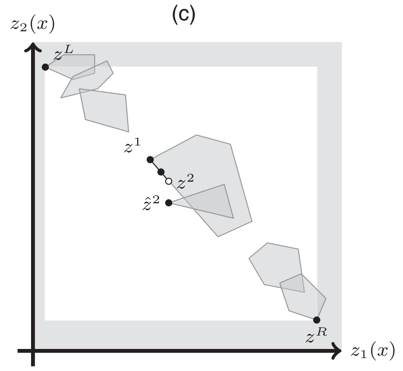
Figure 4. Outer Loop Procedure for the Boxed Line Method When $z_2^* = \mu$



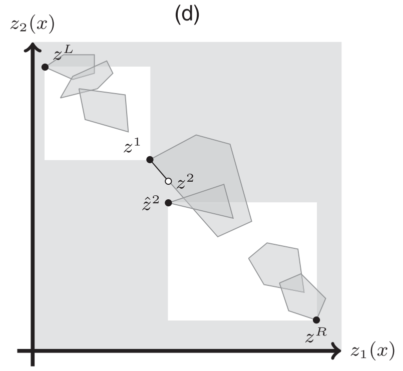
Suppose the box $B(z^L, z^R)$ is chosen from $\mathcal{Q}$.



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



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.



The approximation of the NDF 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).

Let $x^* = (x_I^*, x_C^*) \in \mathcal{X}$ be an optimal solution to (5) that maps to $z^* = z(x^*)$. The first step in the inner loop is to provide an overestimation of the NLS by generating the line segment containing z^* in the slice for x_I^*, restricted to $B(z^L, z^R)$. We discuss the details of the line segment generation subroutine in the Section 3.4. Here we assume that the following is given: the maximal line segment $L(z^1, z^2)$ such that $z^* \in L(z^1, z^2)$, $z_1^L \le z_1^1 \le z_1^2 \le z_1^R, z_2^L \ge z_2^1 \ge z_2^2 \ge z_2^R$, and z^1, z^2, and z^* are all in the slice for x_I^*.

Because $L(z^1, z^2)$ is part of the NDF of the slice problem (i.e., of the slice) for x_I^*, it must be that either (1) $z^1 = z^2$ or (2) $z_1^1 < z_1^2$ and $z_2^1 > z_2^2$. If $z^1 = z^* = z^2$, 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, $z^1 \ne z^2$, and we may define the gradient vector of the line segment $L(z^1, z^2)$, normalized to have unit length, as

$$\vec{w} = (z_2^1 - z_2^2, z_1^2 - z_1^1)/\|(z_2^1 - z_2^2, z_1^2 - z_1^1)\|. \qquad (7)$$

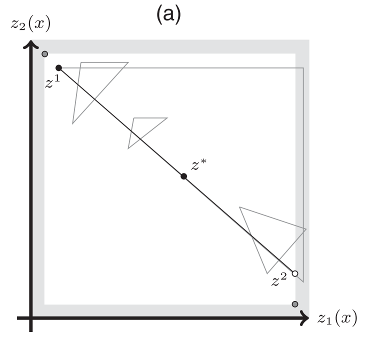
In what follows, all scalarized IPs are scalarized with respect to $\vec{w}$.

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 or z^2 (both their coordinates and their open/closed flag) with any newfound knowledge. For instance, immediately after the line generation subroutine provides $L(z^1, z^2)$, both endpoints are flagged as closed unless the following cases occur: (1) $z_1^R = z_1^2$ and $z_2^R < z_2^2$, illustrated in Figure 5(a), in which case the endpoint z^2 is dominated by z^R, so z^2 cannot be closed and is flagged as open, and (2) $z_2^L = z_2^1$ and $z_1^L < z_1^1$, in which case z^1 is flagged as open.

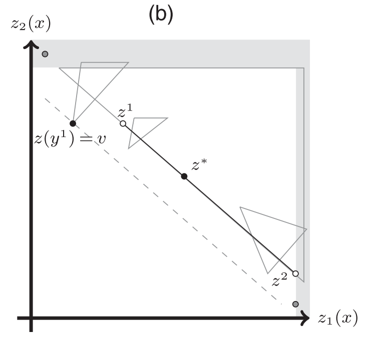
The inner loop is a while loop that continues until the entirety of the current line segment $L(z^1, z^2)$ is nondominated. To check this stopping condition, the scalarized IP

$$z(y^*) = \min\{\vec{w}^T z(x) : z_1(x) \le z_1^2,\ \ z_2(x) \le z_2^1,\ \ x \in \mathcal{X}\} \quad (8)$$

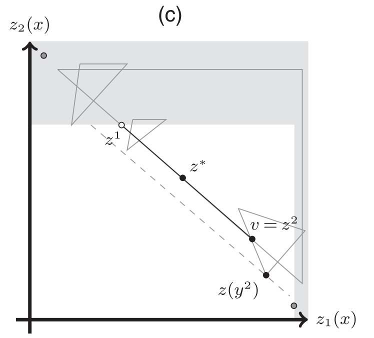
Figure 5. The Inner Loop Takes as Input Nondominated Points z^L, z^R and z^*



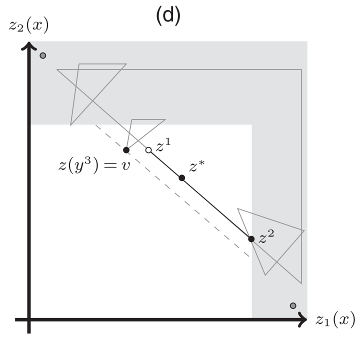
The gray corner points of the white box are z^L and z^R. The darkened line is the approximation of the line segment immediately after line generation subroutine. Note that z^2 is open because it lies directly above z^R: $z_1^2 = z_1^R$ and $z_2^2 > z_2^R$.



Solving (8) (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.



Solving (8) again (within the white region bounded by the previous z^1,z^2) identifies $z(y^2)$. However, v is at the intersection of the slice containing z^* with the line segment, so it becomes the new z^2 which is now closed.



Solving (8) yields $z(y^3)$, then solve for v. 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. $\hat{z}^1 = z(y^3)$.

Note. The output is the (maximal) nondominated line segment containing z^*, $L(z^1, z^2)$, and the nondominated point that dominates any open endpoint.

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 (8) yields an optimal solution y^* with $\vec{w}^T z(y^*) = \vec{w}^T z^*$. Thus $\vec{w}^T z(y^*) = \vec{w}^T z^*$ is an appropriate stopping criterion for the while loop.

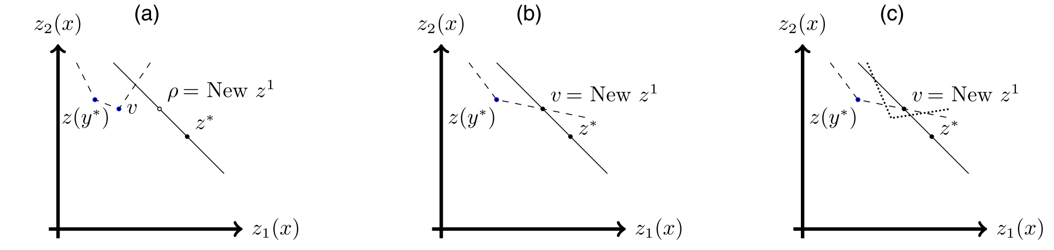
If $\vec{w}^T z(y^*) \neq \vec{w}^T z^*$, it must be that $\vec{w}^T z(y^*) < \vec{w}^T z^*$ because z^* in $L(z^1, z^2)$ ensures that the feasible solution mapping to z^* is feasible for (8). Thus the NDP $z(y^*)$ dominates a portion of the current line segment. There are two cases: either $z(y^*)$ 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 for y_I^* that also dominate part of $L(z^1, z^2)$. We update z^1 so as to exclude all such points within the box by finding the point with integer solution y_I^* that dominates part of $L(z^1, z^2)$ and has a minimal z_2-coordinate. Thus we solve the LP

$$\min\{z_2(x) : \vec{w}^T z(x) \leq \vec{w}^T z^*, \quad x_I = y_I^*, \quad x \in \mathscr{X}\}. \tag{9}$$

Let y' be an optimal solution to (9) and define $v = z(y')$. We make some important observations about the LP defined in (9).

- Minimizing $z_2(x)$ ensures that all points on the slice for y_I^* that can dominate the current line segment are excluded when z_2^1 is updated. The idea is illustrated in Figure 6(a).
- Even though the point v may lie on the line segment $L(z^1, z^2)$, it will never be dominated by any point from $L(z^1, z^2)$ because of the first constraint $\vec{w}^T z(x) \leq \vec{w}^T z^*$. Consider Figure 6(b).
- While the first scalarization returns an NDP $z(y^*)$, the point v may be dominated. Consider Figure 6(c). This issue will be discussed later.

Figure 6. (Color online) Observations About the Construction of Linear Program (9)



(a) The slice for y_I^* is dashed. Minimizing $z_2(x)$ on this slice finds v. Updating the upper endpoint z^1 to the point ρ (as open) will prevent y_I^* recurring when solving (8).

(b) The minimization to find v along the slice for y_I^* cannot surpass $L(z^1, z^2)$ because of the constraint $\vec{w}^T z(x) \le \vec{w}^T z^*$. 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 a point from another slice (dotted).

• Because it is an LP, (9) can be solved efficiently. Furthermore, because the solution y^* is feasible for (9), in implementation, we provide this initial feasible solution to the solver, which yields even greater efficiency.

Next, v is used to update z^1. First, suppose that $v \notin L(z^1, z^2)$. Point $\rho \in L(z^1, z^2)$ such that $\rho_2 = v_2$ is computed, and then z^1 is updated ($z^1 = \rho$) and flagged to be open (see Figure 6(a)). 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 NDP on the slice for y_I^*, as in Figure 7. If so, then the new z^1 will be open; otherwise, it will be closed. To check this, the algorithm solves a second LP

$$\min\{z_1(x):\ z_2(x) = v_2,\ \ x_I = y_I^*,\ \ x \in \mathscr{X}\}. \tag{10}$$

Let y'' be an optimal solution of (10), 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 the algorithm flags z^1 as open; otherwise, it is flagged as closed.

After updating z^1 as described, then, by convexity, it is guaranteed that no point from the slice for y_I^* can again be found. Because $\mathscr{X}$ is bounded, which implies that $\mathscr{X}_I$ 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. First, if $z_2^1 = z_2^L$, then clearly $\hat{z}^1 = z^L$ (and similarly for z^2 and z^R). Second, if $z_2^1 = z_2(y^*)$, then $\hat{z}^1 = z(y^*)$ (see Figure 5(d), 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

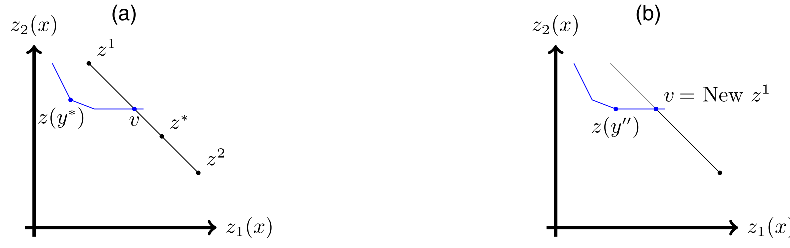
$$\min\{z_1(x) : z_2(x) \le z_2^1,\ x \in \mathscr{X}\} \tag{11}$$

if z^1 is open and by solving the IP

$$\min\{z_2(x) : z_1(x) \le z_1^2,\ x \in \mathscr{X}\} \tag{12}$$

if z^2 is open. If z^1 is open, $\hat{z}^1 = z(\hat{x}^1)$, where $\hat{x}^1$ is an optimal solution to (11), and if z^2 is open, $\hat{z}^2 = z(\hat{x}^2)$, where $\hat{x}^2$ is an optimal solution to (12).

Figure 7. (Color online) The Case That v Lies on the Current Line Segment and Is a Weak Nondominated Point of the Slice Problem for y^*



(a) $z(y^*)$ dominates points from the line segment $L(z^1, z^2)$. The LP (9) 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 (10) yields y'' with $z_1(y'') < v_1$, so the updated z^1 is flagged to be open.

3.4. Line Segment Generation Subroutine

The purpose of the line segment generation subroutine is to provide the inner loop with a single line segment that can be reduced to an NLS simply by updating its endpoints. The subroutine determines an overestimate of the NLS 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 $\min\{(z_1(x), z_2(x)) : x_I = x_I^*, x \in \mathscr{X}\}$. Figure 8 illustrates the three possible cases for z^*: z^*: it may be an isolated point, or it may belong to one or two line segments of the slice. When z^* belongs to two line segments, the subroutine finds the segment to the lower right; that is, it takes $z^1 = z^*$ (see Figure 8(c)).

The line generation subroutine produces three interrelated forms of output: the two endpoints of the line segment, z^1 and z^2, and, as long as $z^1 \neq z^2$, 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. If the gradient vector $\vec{w}$ exists, it is normalized before it is returned. Further details about the subroutine, which is a modified version of dichotomic search (Cohon 1978, Aneja and Nair 1979), can be found in the online supplement.

3.5. Complexity of the Basic Method

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

Consider the BOMIP (1) where $\mathscr{X}$ is nonempty and bounded. Recall that $\mathscr{X}_I$ is the projection of $\mathscr{X}$ onto the set of integer vectors, and S is the index set of $\mathscr{X}_I$. Let $S_{\mathcal{N}}$ be the index set of feasible integer solutions whose slice has nonempty intersection with the NDF $\mathcal{N}$, that is, $S_{\mathcal{N}} = \{s \in S : N^s \cap \mathcal{N} \neq \emptyset\}$. Recall that for all $s \in S_{\mathcal{N}}$, we write $N^s \cap \mathcal{N} = \{L_1^s, L_2^s, \ldots, L_{n(s)}^s\}$, where, for each $i = 1, \ldots, n(s)$, L_i^s is a (distinct, maximal) line segment. 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 $L_i^s \not\subseteq L_j^t$ and $L_j^t \not\subseteq L_i^s$ for all $i = 1, \ldots, n(s)$ and $j = 1, \ldots, n(t)$. We note that in benchmark problems, (point) intersections between distinct slices in the NDF are relatively common, but containment of an NLS from one slice inside that from another is relatively rare. Such containments do not prevent the algorithm from functioning correctly and returning the entire NDF, but the counting of NLSs becomes much more complicated. This assumption does not rule out intersections or even overlap between slices; compare Figure 9, (a) and (b).

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

$$n(Y) := \sum_{s \in S} |N^s \cap \mathcal{N} \cap Y \setminus \{z^L, z^R\}|. \tag{13}$$

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

For simplicity of exposition, we drop the notation for dependence on Y; that is, we use n and g instead of $n(Y)$ and $g(Y)$, respectively. 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 NLSs in the strict interior of Y and g vertical gaps in the NDF.

Figure 8. Three Possible Cases for Finding a Line Segment of the Slice Containing z^*

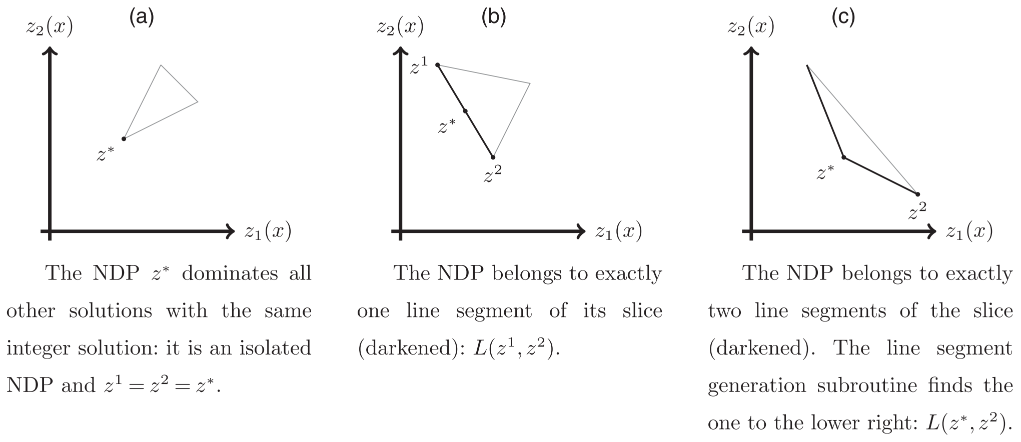
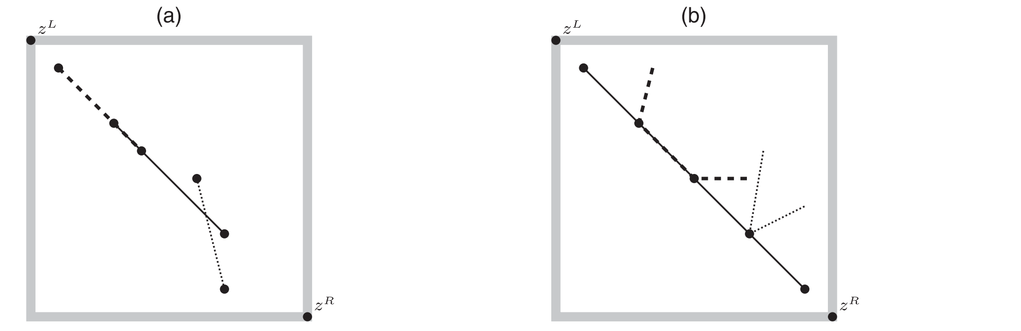

(a)
$z_2(x)$
z^*
$z_1(x)$
The NDP z^* dominates all other solutions with the same integer solution: it is an isolated NDP and $z^1 = z^2 = z^*$.
(b)
$z_2(x)$
z^1
z^*
z^2
$z_1(x)$
The NDP belongs to exactly one line segment of its slice (darkened): $L(z^1, z^2)$.
(c)
$z_2(x)$
z^*
z^2
$z_1(x)$
The NDP belongs to exactly two line segments of the slice (darkened). The line segment generation subroutine finds the one to the lower right: $L(z^*, z^2)$.

Figure 9. Examples of Nondominated Frontiers Where We Assume There Is No Containment Between the Nondominated Line Segment (NLS) of Distinct Slices and How to Count the Number of NLSs n Using (13)



The cases above are "allowed": overlapping and intersecting NLSs from distinct slices, resulting in $n = 3$.

The cases above are assumed to not occur: a NLS from one slice contained within the NLS from another. Without the dashed and dotted slices, $n = 1$ and $g = 2$.

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 $s(n, g)$ 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 $g = 1$, $\ell(0, 1) = 2$ and $s(0, 1) = 0$.

Proof. See the online supplement.

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

Proof. See the online supplement.

As we described while illustrating the basic method, every iteration of the outer loop identifies either an NLS or a vertical gap. The following proposition 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 TSA because it may divide a single NLS into several smaller segments over more than one iteration. Proposition 1 proves 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 $\ell(n, g)$.

Proposition 1. *Suppose that an iteration of the outer loop begins with a region Y having $n \geq 0$ NLSs 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 NLSs in their interior and g', g'' 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. In one iteration of the outer loop, there are two cases for the split line $z_2 = \mu$: either it intersects a vertical gap or it intersects an NLS of the NDF.

In the first case, when the split line intersects a vertical gap of the NDF, 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 NDF(s), so $g' + g'' < g$. However, the total number of NLSs in the interiors of Y' and Y'' is no greater than n, that is, $n' + n'' \leq n$. Thus part 1 of the proposition holds in this case.

In the second case, z^* belongs to some NLS $L(z^1, z^2)$. Thus 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 that the single NLS $L(z^1, z^2)$ is not split. Hence, we must have one fewer NLS in the remaining regions, that is, $n' + n'' < n$. We also have $g' + g'' \leq g$. Thus part 2 of the proposition holds.

Because 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. Thus part 2 of the proposition holds.

Finally, we observe that only the outer loop solves lexicographic IPs. In every iteration of the outer loop, the horizontal split line $z_2 = \mu$ either intersects a vertical gap in the NDF or it intersects an NLS. If the split line intersects an NLS, 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 gap). Also note that Examples 1 and 2 satisfy this upper bound. Thus part 3 of the proposition holds.

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

Proof. See the online supplement.

Because $g \le n+1$, we can define the worst-case number of IP solves as a function of only n, the number of NLSs in the strict interior of Y:

$$\hat{\ell}(n) = \max_{g=1,\dots,n+1} \ell(n,g), \tag{14}$$

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

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

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

Proof of Lemma 2. Assume that there are $n \ge 2$ NLSs. Without loss of generality, we suppose that the number of vertical gaps in the NDF is arbitrary and only consider the case that the split line $z_2 = \mu$ intersects an NLS; otherwise, when the split line intersects a vertical gap, the box is processed in the outer loop without solving any scalarized or single-objective IPs. Once the lexicographic IP (5)—which is not counted for $\hat{s}$—finds NDP z^* on the split line, the inner loop is initiated. Because there are $n-1$ other NLSs in the box, there are at worst $n-1$ scalarized IPs (8) solved in updating the endpoints z^1 and z^2. The nth solve of scalarized IP (8) confirms that the final endpoints on the line segment indeed define an NLS. The worst case is when both z^1 and z^2 are open, $z_2^1 < z_2^L$, and $z_1^2 < z_1^R$ because two additional single-objective IP solves are required to find the NDPs that dominate z^1 and z^2 (a figure illustrating this is given in the online supplement). 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 $\hat{s}(n) = n + 2 + \max_{i=0,\dots,n-1}\{\hat{s}(i) + \hat{s}(n-1-i)\}$.

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

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

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

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

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

Theorem 1. *For all $n \ge 1$, $\hat{s}(n) = n(n+1)/2 + 2(n-1)$.*

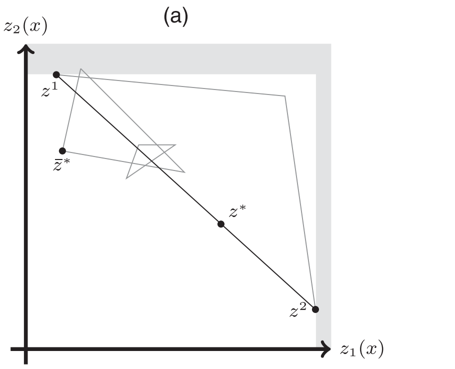
Proof. This follows by induction on n and Lemma 2; see the online supplement for details.

4. Recursive Method

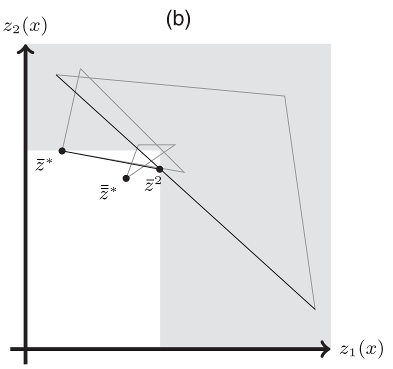
There is a natural way to make the BLM a recursive method that has substantially better worst-case complexity. The fundamental concept is to transform the inner loop into a recursive procedure that either confirms that a line segment is nondominated or identifies an NDP that dominates a portion of it. Whenever such an NDP is found, the inner loop is called recursively to find an NLS containing it. 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 10.

The initialization stage remains the same, and the outer loop follows the same procedure as the basic method before calling the inner loop; that is, it selects boxes from the queue, chooses a split line $z_2 = \mu$, solves lexicographic minimization (5), and follows the BBM procedure when the NDP found is below 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, that is, $z_2^* = \mu$, the outer loop then calls the recursive inner loop for the first time, which we call *depth level 0*.

Figure 10. The Recursive Inner Loop Applied to a Nondominated Point (NDP) $\bar{z}^*$ Returns Nondominated Line Segments $L(z^1, z^2)$ and $L(\bar{z}^*, \bar{z}^2)$ and the Isolated NDP $\bar{\bar{z}}^*$

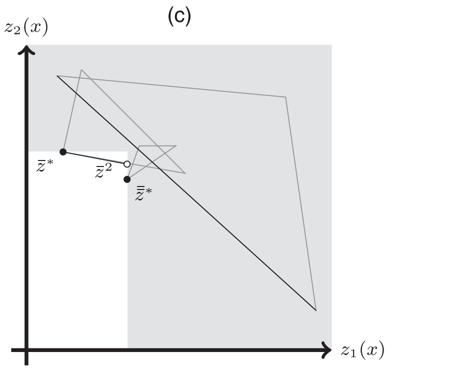


Level 0: The original NDP is z^*, and line generation returns $L_0 = L(z^1, z^2)$, which is not trimmed. Solving a scalarized IP over the white region identifies NDP $\bar{z}^*$.

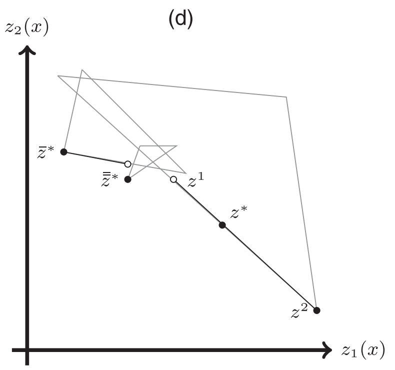


Level 1: The line generated for $\bar{z}^*$ is trimmed by its intersection with L_0, resulting in $L_1 = L(\bar{z}^*, \bar{z}^2)$ with $\bar{z}^2$ closed. NDP $\bar{\bar{z}}^*$ solves the next scalarized IP.

Level 2: Returns isolated NDP $\bar{\bar{z}}^*$.



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.



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.

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 NLSs $\mathcal{M}$, 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 provides the line segment $L(z^1, z^2)$ and its gradient vector $\vec{w}$.

The 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 because $L(z^1, z^2)$ 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, that is, 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 same scalarized IP with respect to $\vec{w}$ as in the basic method, that is, (8), including the conditionally strict inequalities. Say the optimal solution is y^*. There are two cases. In the first case, if $L(z^1, z^2)$ is nondominated, then $z(y^*)$ lies on the line segment; that is, $\vec{w}^T z(y^*) = \vec{w}^T z^*$. This is the stopping criterion for the while loop. When it is satisfied, the recursive function terminates and returns the NLS, together with all others it found, stored in $\mathcal{M}$. In the second, more interesting case, $\vec{w}^T z(y^*) < \vec{w}^T z^*$, that is, when a point from $L(z^1, z^2)$ is dominated by $z(y^*)$. In this case, the function recurses and calls itself at the next depth level. Assume that $z(y^*)$ 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:

- $B(\chi, z^*)$, the new box, where χ is the NDP from $\mathcal{M} \cup \{z^L\}$ 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 $B(\chi, z^*)$ as the new box prevents rediscovery 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 $\mathcal{M}$ is empty, $\chi = z^L$.

After all deeper levels of recursive processing are complete, a set of NLSs $\mathcal{M}$ 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 (9), it chooses the NDP from $\mathcal{M}$ that is the nearest to and strictly to the left of z^*. That is, it chooses

$$\sigma = \operatorname{argmin}\{z_2 : z \in \mathcal{M}, z_1 < z_1^*\}. \tag{16}$$

The coordinate σ_2 gives the updated z_2^1, and the updated z_1^1 value is calculated to ensure that z^1 remains on its original line segment. Whether it is closed or open is determined by whether σ lies on the line segment. Because σ 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 (8). If the updated line segment is nondominated, then the stopping criterion is met, and the line segment $L(z^1, z^2)$ is returned, along with any others collected in $\mathcal{M}$. Otherwise, the while loop continues.

The recursive inner loop accumulates all NLSs identified throughout all depths of recursion. This accumulated set $\mathcal{M}$ is returned to the outer loop, which updates $\tilde{\mathcal{N}}$ and $\mathcal{Q}$ 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 11(a). Suppose that z^0 was 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 (8) 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 developed a simple routine to detect such intersections and update the child's line segment so as to prevent cycling. This is done by solving a linear system and updating one endpoint of the child's line segment, as necessary; see the online supplement for details.

The line trimming process must be repeated for all line segments in $\mathcal{L}$ that have been inherited by the current call. For instance, observe that in Figure 11(b), 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

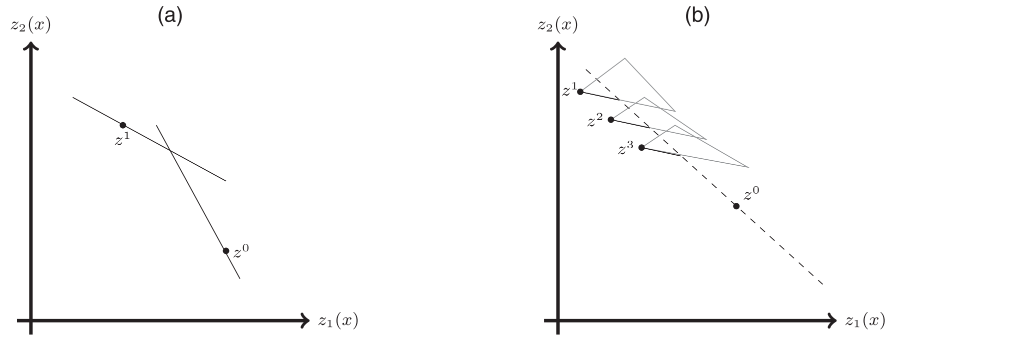
The only major modification in the outer loop is the handling of the output from the recursive inner loop, which may return more than one continuous portion of the NDF (see Figure 12). Therefore, we must allow the outer loop to add more than one NLS to the NDF and more than two boxes to the queue, as necessary.

The latter requires a simple check of neighboring NLSs. 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 12, 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 the 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.

Figure 11. The Line Segment Trimming Subroutine Prevents the Recursive Method from Cycling



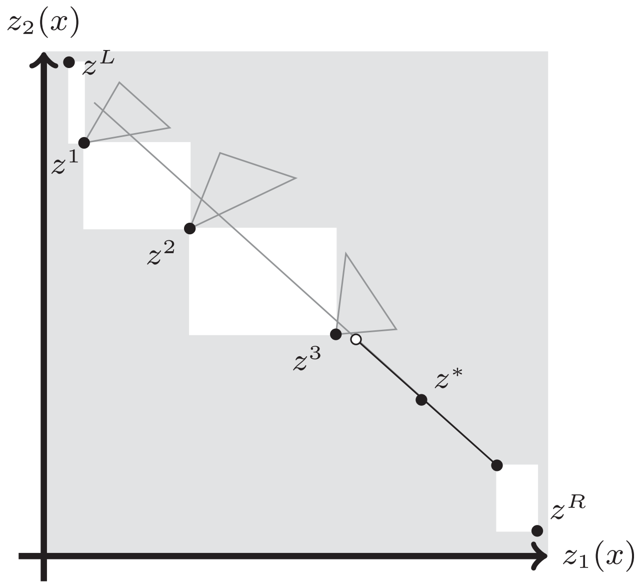
Without the line segment trimming, the recursive method would cycle between finding z^0 and z^1.

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.

The initialization stage and most of the functionality of the outer loop are unchanged; that is, in every iteration, the outer loop solves one lexicographic IP to identify an NLS, or it solves two lexicographic IPs to identify a vertical gap. What has changed is the updating procedure for the NDF 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; that is, $\hat{\ell}_R(n) = \hat{\ell}(n) = 3n + 2$. However, we must reconsider an upper bound for the number of scalarized IP solves.

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

Figure 12. The Recursive Variant May Return More Than One Continuous Portion of the Nondominated Frontier (e.g., the Nondominated Line Segment Containing z^* and Three Isolated Nondominated Points in the Example Above), in Which Case the Algorithm Adds More Than Two Boxes to the Queue (in White)



Proof of Proposition 2. Let z^L, z^R be the corner points for a box with $n \geq 1$ NLSs. Assume that the NDF in the box contains no vertical gaps. This assumption can be made without loss of generality because vertical gaps in the NDF are only discovered in the outer loop (by a lexicographic IP solve that yields an NDP not on the split line). They do not change the number of scalarized IPs that need to be solved.

Because 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 scalarized IP is solved per NLS in the NDF. 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, no more than $n - 1$ type II scalarized IPs will be required to discover all the NLSs (the first NLS must be found by a lexicographic IP). Because all n NLSs require solving 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. Same Integer Solution Enhancement

We propose an enhancement to the basic method (given in Section 3) that has the potential to improve its computational performance. The enhancement 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 call this the *same integer solution* (SIS) variant.

This enhancement was motivated by a common structure encountered in the NDFs of benchmark instances used in previous work on BOMIP algorithms (Boland et al. 2015b, Soylu and Yıldız 2016). These instances have NDFs consisting of a relatively small number of continuous sections, generated by relatively few slices, each having many small NLSs. Figure 13 gives plots of two such instances. In one, only five slices contribute to the NDF, which consists of only four 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.

During the BLM's processing of these instances, when both corner points of a box have the SIS, that is, 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 NDF between z^L and z^R is a subset of the slice for x_I^*. The SIS variant exploits this observation. In the case that the slice problem does yield the NDF in the current box, the SIS variant 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 NDF, 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 slice 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 (integer part of the solution is fixed) in criterion space.

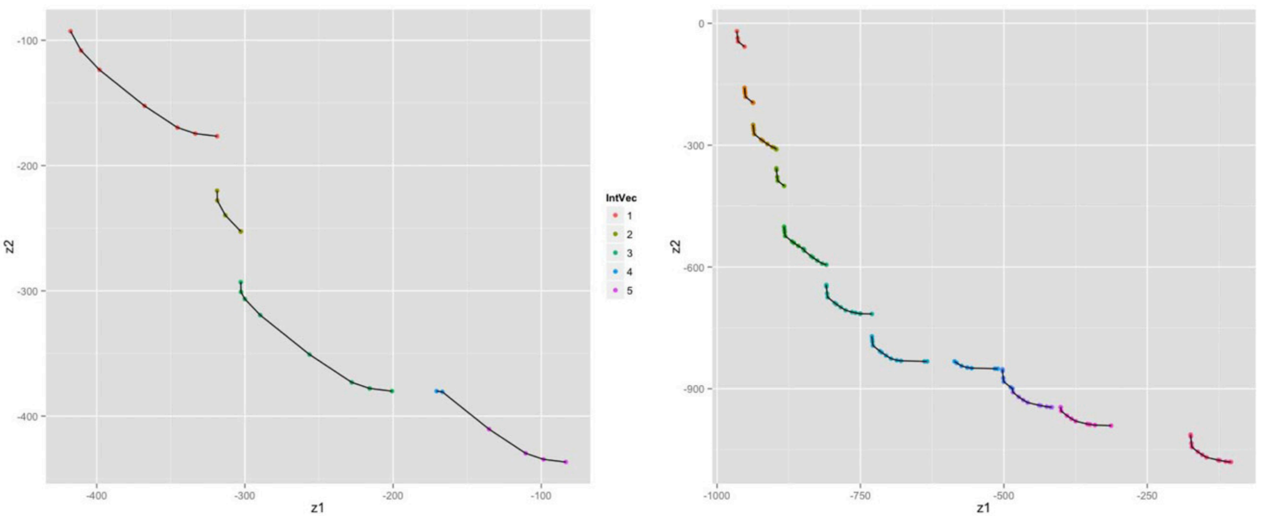
Let $\vec{w}$ denote the gradient vector of $L(z^L, z^R)$. The following scalarized IP with respect to $\vec{w}$ includes a no-good constraint $x_I \neq x_I^*$, which makes all points from the slice for x_I^* infeasible:

$$\min\{\vec{w}^T z(x) : z_1(x) \le z_1^R,\ z_2(x) \le z_2^L,\ x_I \neq x_I^*,\ x \in \mathscr{X}\}. \tag{17}$$

We assume that the no-good constraint is implemented linearly in the usual way. Let y^* be an optimal solution to (17). 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 $L(z^L, z^R)$, that is, $\vec{w}^T z^L \le \vec{w}^T z(y^*)$, then $z(y^*)$ must be dominated by a point from the slice for x_I^*, as shown in Figure 14(a). Furthermore, the entire NDF 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 (Cohon 1978, Aneja and Nair 1979), which solves a series of LPs.[3] Therefore, if $z(y^*)$ is on or above the line segment $L(z^L, z^R)$, the SIS variant will generate the entire NDF 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 BLM. In the same scenario with n such NLSs in the NDF (all from the same slice), it would solve n lexicographic IPs and n scalarized IPs. Thus the SIS variant has the potential to provide significant savings in the number of IPs solved.

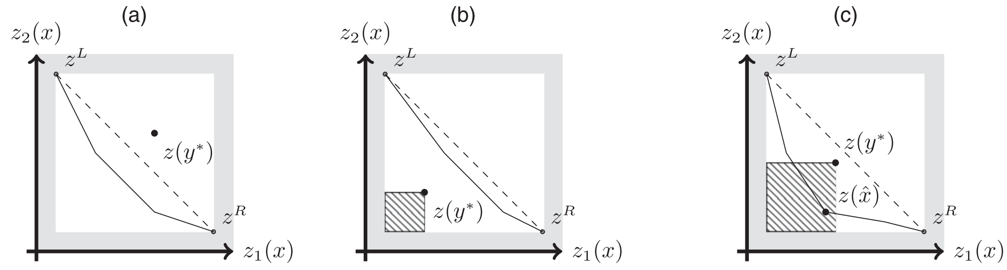
Now consider the other case, that $z(y^*)$ is below the line segment $L(z^L, z^R)$, that is, that $\vec{w}^T z(y^*) < \vec{w}^T z^L$. There are two subcases: either $z(y^*)$ is an NDP, or $z(y^*)$ is dominated by some point on the (so far unknown) slice for x_I^*. These subcases are illustrated in Figure 14, (b) and (c), respectively.

Figure 13. (Color online) The Nondominated Frontiers for Two Benchmark Instances



Notes. The keys indicate the integer vector associated with each nondominated point. Notice the similar structure in both: each integer vector contributes several small and continuous nondominated line segments to the nondominated frontier.

Figure 14. The Same Integer Solution Variant Applied When z^L and z^R Are Both Generated by Integer Solution x_I^*



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

If $z(y^*)$ is found below $L(z^L, z^R)$, an LP over the hatched 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 $z^* = z(y^*)$.

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

Note. The scalarization (17) 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).

When $z(y^*)$ is below $L(z^L, z^R)$, the SIS variant solves the LP

$$\min\{\vec{w}^T z(x) : z_1(x) \le z_1(y^*),\ z_2(x) \le z_2(y^*),\ x_I = x_I^*,\ x \in \mathscr{X}\}. \qquad (18)$$

If LP (18) is infeasible, then $z(y^*)$ is an NDP. This follows because $z_i(x) \le z_i(y^*)$ for $i = 1, 2$ and $x \in \mathscr{X}$ implies, by the definition of y^*, that either $z(x) = z(y^*)$ or $x_I = x_I^*$. The latter would imply that x_I is feasible for LP (18), which is impossible. Hence, $z(x) = z(y^*)$; $z(y^*)$ must be an NDP. Otherwise, for similar reasons, any feasible solution of LP (18), $\hat{x}$ 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 variant 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). By contrast, the basic BLM would solve one lexicographic IP before calling the inner loop, with similar output. Therefore, the computational effort (and return) is comparable, so there is not much benefit from the SIS variant in this case.

Computational experiments with the SIS variant 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 BLM to provide output (defined more precisely in Section 6.1). Note that because the choice of ϵ impacts (the accuracy of) the NDF generated by an algorithm, it also impacts the time required by the algorithm to find the NDF. In Section 6.2, we show that the NDFs generated for the same instance for different values of ϵ can vary drastically.

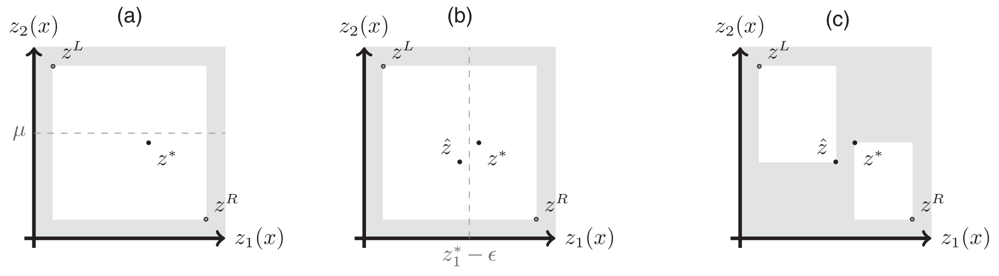
Single-objective IP solvers use tolerances as well, for example, feasibility and optimality tolerances, and their choice also impacts the performance. In fact, certain choices of IP solver tolerances may lead to unexpected behavior in the algorithm. An illustration is provided in Figure 15. This is just one example of the 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, that is, $z_1^1 \le z_1(x) \le z_1^2$ and $z_2^2 \le z_2(x) \le z_2^1$, or by just two constraints, that is, $z_1(x) \le z_1^2$ and $z_2(x) \le z_2^1$ (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 an ϵ-approximation of the NDF $\mathcal{N}$ or simply an 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 $\|z - \bar{z}\|_2 \le \epsilon$, and (2) for all $\bar{z} \in \mathcal{N}_\epsilon$, there exists $z \in \mathcal{N}$ with $\|z - \bar{z}\|_2 \le \epsilon$.[4] Note that there

Figure 15. If Mixed Integer Linear Program (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



In $B(z^L, z^R)$, the first lexicographic IP finds NDP z^* below the split line $z_2 = \mu$.

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^*.

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.

may be many distinct sets that satisfy the conditions for being an ϵ-frontier. The BLM (when run to completion) produces an ϵ-frontier for any given BOMIP having a nonempty and bounded feasible set. To see how ϵ is used in the method to produce the ϵ-frontier, see the algorithms in the online supplement.

This definition of an ϵ-frontier motivated specific design choices in our implementation. For example, when the inner loop solves scalarization IPs, for example, scalarized IP (8), to find NDPs that dominate a line segment, our definition implies that we are only interested in NDPs whose distances 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 optimal 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, that is, 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 (Boland et al. 2015b, Soylu and Yıldız 2016) are based on the scheme proposed by Mavrotas and Diakoulaki (1998). The NDFs for the 20 instances, ranging in size from 20 to 320 decision variables, have a structure that was described in Section 5. Much of the NDF consists of continuous line segments from the same slice (see Figure 13). This structure not only poses numerical challenges when the line segments are extremely small, that is, when their lengths are smaller than ϵ, it also means that changes in the value of ϵ result in drastically different NDFs being generated by our algorithms, as shown in Table 1.

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 an algorithm, such as the TSA, may output several small line segments whose union is equivalent to a single line segment output by another algorithm, such as the εTCM. This has motivated us to generate new instances for which the exact NDF is known and the line segments in the frontier all have length greater than a prespecified value, for example, 10^{-4}.

7. Instance Generation

We now provide a method for generating a BOMIP having an NDF controlled by parameters and known a priori. In doing so, we provide an approach to reverse engineer a BOMIP from the slices.[5]

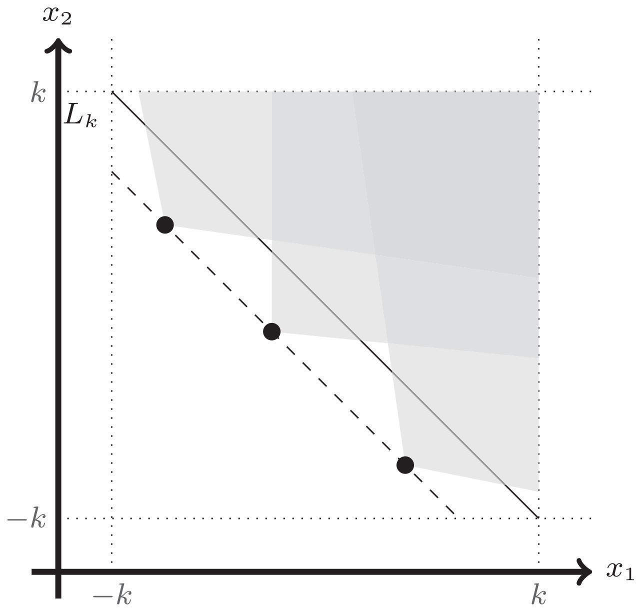
Each instance's NDF includes some sections of the line segment $L_k = \{(x_1, x_2) \in \mathbb{R}^2 : x_1 + x_2 = 0, -k \leq x_i \leq k \text{ for } i = 1, 2\}$, where $k \in (0, \infty)$ is a parameter. This line segment will be one slice in the instance. The instance has π other slices, where π is 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_k but shifted vertically down. The width of the cone is randomly chosen in a controlled manner. Figure 16 illustrates this structure, showing the image of the feasible set in criterion space for four

Table 1. Number of Nondominated Line Segments Found for Different Values ϵ

Instance	10^{-3}	10^{-4}	10^{-5}	10^{-6}
1	2,570	2,752	2,783	2,786
2	2,643	2,916	2,958	2,970
3	2,578	2,723	2,750	2,753
4	5,608	6,121	6,164	6,174
5	2,824	3,054	3,096	3,096

Note. All five instances are from the C160 class of instances and are solved by the basic method.

Figure 16. An Example of a Generated Randomized-Cone-Width Instance with Four Slices, Including L_k and the Boundary of Three Cones



slices: L_k and three pointed cones. All instances are constructed to have the property that no two cones overlap on or below L_k. This means that L_k alternates between a section that is part of the NDF and a section dominated by one cone.

Details of a method to generate two different classes of instances having the structure described earlier are provided in the online supplement. One class, the fixed-cone-width instances, is preferred for assessment of the accuracy of BOMIP algorithms because the NDF is known exactly a priori. The other class, the randomized-cone-width instances, does not have known NDFs, and this class of instances is more difficult to solve in practice because of the high frequency of intersecting slices in the NDF. All instances are carefully designed to ensure that each NLS has length of at least ϵ. We also force a proportion of the cones in a randomized-cone-width instance to be orthogonal;[6] this induces open endpoints.

Given π, the resulting BOMIP has a number of variables and a number of constraints that are linear in π. The NDFs associated with the BOMIP will have no more than $3\pi + 1$ line segments ($\pi + 1$ from L_k and at most two per cone), including open endpoints induced by any orthogonal cones and closed endpoints induced by intersecting slices.

These 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 BLM on the following sets of instances:

- The 10 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 $\pi = 5{,}000$ and five obtained by setting $\pi = 7{,}500$, 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 2.3-GHz with 50 GB of random-access memory running Red Hat Enterprise Linux Server 7.4.

All variations of BLM use tolerances $\epsilon = 10^{-5}$ and CPLEX tolerances 10^{-7} for historical instances and tolerances $\epsilon = 10^{-4}$ and CPLEX tolerances 10^{-6} for the new instances. The reason for using greater accuracy for the historical instances is that many of the NDFs have very small line segments as well as nearly horizontal or vertical line segments, which makes the accurate calculation of the gradients critical.

We structure the discussion of the computational experiments as follows. In Section 8.1, we present a comparison of the variants of BLM, that is, basic, SIS, and recursive, and in Section 8.2, we compare (variants of) BLM with the TSA and the εTCM.

8.1. Comparison of BLM Variants

Table 2 shows the results for the largest historical instances, that is, the class C320 instances (results for the next-largest instances, i.e., the class C160 instances, can be found in the online supplement). Table 3 summarizes the results for the new instances by presenting the mean of the performance metrics (individual results can be found in the online supplement). For each combination of algorithm variant and instance, we provide the following statistics: number of points output by the algorithm (nNDP; described in more detail later), number of different integer part solutions (nIPF; i.e., the number of different slices appearing in the NDF), total time in seconds to discover the NDF (TT), total time in seconds spent in IP solves (IPT), total time in seconds spent in LP solves (LPT), total number of IPs solved (nIP), total number of lexicographic IPs (nLex), total number of single-objective IPs solved to find the NDP that dominates an open endpoint, that is, (11) and (12) (nMin), total number of scalarized IPs, for example, (8) (nScal), total number of IPs with the no-good constraint (nGood), total number of LPs solved (nLP), total number of boxes processed (nRec), total number of boxes processed where the corner points are generated by the SIS (nSIS), and finally, number of boxes with z^* on the horizontal split line (nZL). When not applicable, an entry in the tables is marked with "N/A."

Table 2. Comparison of the Different Algorithms for Historical Instances, Class C320

Algorithm	Instance	nNDP	nIPF	TT	IPT	LPT	nIP	nLex	nMin	nScal	nGood	nLP	nBox	nSIS	nZL
Basic	21	16,850	294	37,665.5	33,307.6	4,339.7	53,514	17,790	40	17,894	N/A	171,370	17,803	15,926	17,766
	22	19,778	410	42,045.3	36,927.4	5,095.7	61,766	20,540	23	20,663	N/A	191,611	20,510	18,074	20,476
	23	17,319	343	38,995.1	34,570.3	4,409.4	53,859	17,895	26	18,043	N/A	171,514	17,884	15,776	17,871
	24	19,898	460	46,967.1	41,632.8	5,312.7	62,338	20,706	34	20,892	N/A	200,619	20,696	17,867	20,682
	25	13,682	337	24,450.1	21,268.9	3,171.0	42,196	14,024	27	14,121	N/A	130,934	13,994	12,130	13,964
Average		17,505.4	368.8	38,024.6	33,541.4	4,465.7	54,734.6	18,191	30	18,322.6	N/A	173,209.6	18,177.4	15,954.6	18,151.8
Same integer solution	21	15,699	294	6,106.2	4,890.5	1,211.5	6,598	1,741	40	1,849	1,227	43,443	2,956	1,227	1,722
	22	18,840	410	7,788.7	6,297.5	1,485.3	8,613	2,287	23	2,421	1,595	52,662	3,850	1,595	2,233
	23	16,449	343	6,977.4	5,607.1	1,366.8	7,459	1,982	26	2,131	1,338	46,627	3,309	1,338	1,961
	24	18,546	460	9,535.6	7,899.2	1,630.3	10,070	2,672	34	2,884	1,808	56,219	4,468	1,808	2,679
	25	13,239	337	5,329.1	4,296.6	1,029.2	6,578	1,753	26	1,853	1,193	37,911	2,916	1,193	1,700
Average		16,554.6	368.8	7,147.4	5,798.2	1,344.6	7,863.6	2,087	29.8	2,227.6	1,432.2	47,372.4	3,499.8	1,432.2	2,059
Recursive	21	16,831	294	37,201.4	32,767.3	4,417.0	52,297	16,979	N/A	18,339	N/A	170,040	16,971	15,611	16,955
	22	19,763	410	41,650	36,417.7	5,210.1	60,312	19,600	N/A	21,112	N/A	189,879	19,568	17,830	19,536
	23	17,315	343	38,684.6	34,149.4	4,521.7	52,585	17,042	N/A	18,501	N/A	170,082	17,030	15,546	17,018
	24	19,890	460	46,196.8	40,725.8	5,449.5	60,571	19,576	N/A	21,419	N/A	198,351	19,566	17,509	19,552
	25	13,667	337	24,635	21,314.5	3,310.4	40,899	13,143	N/A	14,613	N/A	129,295	13,113	11,834	13,083
Average		17,493.2	368.8	37,673.6	33,074.9	4,581.7	53,332.8	17,268	N/A	18,796.8	N/A	171,529.4	17,249.6	15,666	17,228.8

Notes. Statistics reported: nNDP, number of points output by the algorithm; nIPF, number of different integer part solutions; TT, total time to discover the NDF; IPT, total time spent in IP solves; LPT, total time spent in liner program (LP) solves; nIP, total number of mixed integer linear programs (IPs) solved; nLex, total number of lexicographic IPs; nMin, total number of single-objective IPs solved to find the nondominated point (NDP) that dominates an open endpoint; nScal, total number of scalarized IPs; nGood, total number of IPs with the no-good constraint; nLP, total number of LPs solved; nRec, total number of boxes processed; nSIS, total number of boxes processed where the corner points are generated by the same integer solution; nZL, number of boxes with z^* on the horizontal split line. Times are reported in seconds. N/A, not applicable.

Table 3. Comparison of the Different Algorithms for Generated Instances with n = 5,000 and n = 75,000

Algorithm	N	nNDP	nIPF	TT	IPT	LPT	nIP	nLex	nMin	nScal	nGood	nLP	nBox	nSIS	nZL
Basic	5,000	15,002.0	5,001	3,924.3	1,761.6	1,846.8	51,786.4	14,586.6	466.4	22,146.8	N/A	104,131	14,546.4	3,175.0	14,502.6
Same integer solution	5,000	15,002.0	5,001	4,522.0	2,245.0	1,954.0	53,575.4	16,191.2	379.6	20,732.6	80.8	112,810.4	16,251.4	80.8	16,073.2
Recursive	5,000	15,002.4	5,001	2,823.6	1,341.8	1,187.8	29,628.2	3,633.6	N/A	22,361	N/A	72,896.4	3,612.0	1.0	3,590.4
Basic	7,500	22,502.0	7,501	8,881.4	4,109.9	3,914.8	81,398.0	21,846.8	706.4	36,998	N/A	157,289.4	21,795.4	8,148	21,740.0
Same integer solution	7,500	22,502.0	7,501	12,060.1	7,146.5	4,017.5	79,738.8	24,021.0	606.0	30,979.8	111.0	168,228.0	24,098.8	111.0	23,888.8
Recursive	7,500	22,502.6	7,501	6,346.7	3,002.0	2,538.4	44,452.4	5,272.8	N/A	33,906.8	N/A	109,628.6	5,240.4	1.2	5,208.0

Notes. Statistics reported: nNDP, number of points output by the algorithm; nIPF, number of different integer part solutions; TT, total time to discover the NDF; IPT, total time spent in IP solves; LPT, total time spent in liner program (LP) solves; nIP, total number of mixed integer linear programs (IPs) solved; nLex, total number of lexicographic IPs; nMin, total number of single-objective IPs solved to find the nondominated point (NDP) that dominates an open endpoint; nScal, total number of scalarized IPs; nGood, total number of IPs with the no-good constraint; nLP, total number of LPs solved; nRec, total number of boxes processed; nSIS, total number of boxes processed where the corner points are generated by the same integer solution; nZL, number of boxes with z^* on the horizontal split line. All metrics are averaged over five instances. Times are reported in seconds. N/A, not applicable.

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

The value reported in the "nNDP" column for a given instance ideally should 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 × nLex + nMin + nScal + nGood.

First, we discuss the results from Table 2, starting with the basic variant. As shown in Figure 13, the NDF of a historical instance is characterized by continuous portions of many small NLSs from the same slice. In fact, the basic variant of the BLM reports an average ratio of 47.5 NDPs per integer solution (i.e., nNDP/nIPF). 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 ≈ IPT + LPT). The average percentage of the total time spent solving IPs is 88.2%. We note that the number of lexicographic IPs is quite close to the number of NDPs. This is to be expected because in the basic variant of the 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 because the majority of 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 98.1% of the time (on average). The number of boxes processed is also quite close to the number of NDPs because 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: 87.8% of the boxes (on average). 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: 99.9% of the NDPs (on average). 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 SIS variant, as expected, greatly improves on the basic variant. Given a box with corner points having the SIS, the SIS variant is able to prove that the slice for the integer solution 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 81.2% for total time (on average) and 85.6% for number of IP solved (on average). This is the best variant of the 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 gradient for a line segment is not very accurate, then the LPs 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 NDF inside the box belongs to a single slice, the NDPs of that slice problem are found by solving LPs without the need to calculate the individual gradient of each line segment.

The recursive variant does not improve on the basic variant because it does not recurse very often (the maximum recorded depth was three for all instances), 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 three NDPs per integer solution for both sets (n = 5,000 and n = 7,500). This is expected due to their structure. The percentage of time spent solving IPs is 44.9% for set n = 5,000 and 46.3% for set n = 7,500. 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 an NLS. The number of scalarized IPs is approximately three per two NDPs. This shows that more scalarized IPs per NDP are solved in the new instances compared with the historical instances. This is due to the characterization of the NDF, 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 = 5,000 and 59.3% of the time (on average) for set n = 7,500. A reasonable number of the boxes processed have corner points with the SIS, an average of 21.8% for set n = 5,000 and 37.4% for set n = 7,500. This is not surprising given the way the new instances are constructed, that is, with line segment L_k extending across the NDF and contributing many distinct NLSs. However, unlike the historical instances, the NDF within these boxes is less likely to be generated entirely by that single integer solution.

As expected, the SIS variant does not improve on 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 SIS of the line segment L_k, 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 three NDPs). Second, we found that solving scalarized IPs in which the objective function has the same gradient as the line segment L_k can require excessive amounts of time, most likely because there are multiple optimal solutions to the scalarized IP (given the way the corner points of the cones are generated). The very first scalarized IP solved can 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 with 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 the set n = 5,000 and 71.5% of the total time and 55.6% of the number of IPs for the set n = 7,500. 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 = 5,000 and from 15 to 20 for instances in the set n = 7,500. 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, because the recursive inner loop does not solve as many LPs (e.g., it does not solve (9) to find v), the total number of LPs drops noticeably compared with the basic variant.

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 nontrivial, in practice, to characterize an optimal solution, that is, the NDF. An NDF 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 NDF of a multiobjective integer program, as we have seen in Section 6, can have a noticeable effect on the resulting NDF.

Unfortunately, the use of tolerances in algorithms for finding the NDF of a multiobjective 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 NDF 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 TSA and the εTCM both made the source code of the implementation of their algorithms available to us. Thus, a comparison of the performance of our proposed algorithms with 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, that is, 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 the εTCM reads the instances correctly but terminates after the first integer program is solved.

Furthermore, the implementation of the TSA that we had access to uses a relative tolerance and, as a result, only finds an approximation of the NDF. More specifically, averaged over the instances in the sets C160 and C320, the 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, the 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 solution times.[7]

The implementation of the εTCM that we had access to produced almost identical NDFs, differing by only a few NDPs, but for a few historical instances, it failed to produce an NDF (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. However, because we felt it was important to perform a thorough and fair comparison on all instances, new as well as historical, we implemented our own version of the εTCM, using, whenever possible, data structures and subroutines common to the BLM. We validated our implementation of the εTCM 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, that is, for which the original implementation ran on our platform.

Our overall comparison of algorithms can be found in Table 4, which reports on the performance of the SIS variant of the BLM and the εTCM on the historical instances and the recursive variant of the BLM and the εTCM on the new instances.

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

Even though the recursive variant of the BLM solves more IPs than the εTCM and far more LPs than the εTCM, on the new instances, it still ends up having significantly faster solution times, 2,823.6 seconds versus 12,156.3 seconds, respectively, averaged over the $n =$ 5,000 instances, and 6,346.7 seconds versus 37,922.2 seconds, respectively, averaged over the $n = 7{,}500$ instances.

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

8.2.1. Comments on Recently Published Algorithms for Solving BOMIPs.

The growing interest in solving BOMIPs is illustrated by two very recent papers (both published while this paper was under review): Fattahi and Turkay (2018) introduce an algorithm, called *one-direction search* (ODS), based on ideas similar to the εTCM, and Soylu (2018) introduces an algorithm, called *search-and-remove* (SaR), that cleverly combines dichotomic search, solving slice problems, and no-good constraints. For the sake of completeness, we provide some comments on the computational results reported in these papers.

Fattahi and Turkay (2018) and Soylu (2018) provide a comparison with published results for the TSA on historical instances, classes C160 and C320, acknowledging the challenges associated with such a comparison because their results were obtained on different hardware platforms and using different versions of CPLEX.

The results for the ODS with relative error $\xi = 10e^{-5}$ (Fattahi and Turkay 2018) show that the ODS takes more time to produce the NDF for class C160 instances and about the same time or slightly less time

Table 4. Comparison of the εTCM and the BLM (SIS Variant for Historical and Recursive Variant for New Instances)

	εTCM							BLM						
Instance	nNDP	nIPF	TT	IPT	LPT	nIP	nLP	nNDP	nIPF	TT	IPT	LPT	nIP	nLP
16	2,762	115	216.6	145.8	70.3	2,870	10,520	2,764	115	206.0	148.6	57.1	1,934	8,287
17	2,949	110	324.9	194.1	130.2	3,054	16,205	2,952	110	257.7	185.6	71.7	1,960	8,915
18	2,739	101	239.3	156.4	82.4	2,834	11,257	2,738	101	232.8	170.5	61.9	1,909	8,172
19	6,151	181	674.1	494.9	177.1	6,319	23,086	6,166	181	577.1	433.1	143.1	3,805	18,212
20	3,125	100	273.0	202.4	69.9	3,216	9,080	3,122	100	216.1	148.8	66.9	1,748	8,698
Average	3,545.2	121.4	345.6	238.7	106.0	3,658.6	14,029.6	3,548.4	121.4	297.9	217.3	80.2	2,271.2	10,456.8
21	15,636	295	8,836.8	6,659.8	2,165.8	15,923	86,957	15,699	294	6,106.2	4,890.5	1,211.5	6,598	43,443
22	18,825	410	11,671.6	9,055.6	2,598.7	19,205	98,134	18,840	410	7,788.7	6,297.5	1,485.3	8,613	52,662
23	16,420	343	10,437.3	7,743.2	2,682.4	16,752	102,411	16,449	343	6,977.4	5,607.1	1,366.8	7,459	46,627
24	18,471	457	13,028.6	10,187.5	2,825.8	18,968	101,218	18,546	460	9,535.6	7,899.2	1,630.3	10,070	56,219
25	13,216	337	7,799.5	5,698.3	2,092.7	13,518	79,133	13,239	337	5,329.1	4,296.6	1,029.2	6,578	37,911
Average	16,513.6	368.4	10,354.8	7,868.9	2,473.1	16,873.2	93,570.6	16,554.6	368.8	7,147.4	5,798.2	1,344.6	7,863.6	47,372.4
5,000.A	15,002	5,001	12,152.0	11,513.9	337.8	25,229	24,345	15,002	5,001	2,861.8	1,383.9	1,184.6	29,711	72,937
5,000.B	15,002	5,001	12,185.2	11,565.4	337.3	25,281	24,201	15,002	5,001	2,788.4	1,301.3	1,192.8	29,587	72,826
5,000.C	15,002	5,001	12,218.9	11,586.8	338.4	25,292	24,156	15,004	5,001	2,827.3	1,350.4	1,184.6	29,538	72,824
5,000.D	15,002	5,001	12,196.0	11,563.4	338.2	25,256	24,272	15,002	5,001	2,788.5	1,306.2	1,187.8	29,664	72,986
5,000.E	15,002	5,001	12,029.3	11,415.1	334.1	25,263	24,243	15,002	5,001	2,851.8	1,367.3	1,189.4	29,641	72,909
Average	15,002	5,001	12,156.3	11,528.9	337.2	25,264.2	24,243.4	15,002.4	5,001	2,823.6	1,341.8	1,187.8	29,628.2	72,896.4
7,500.A	22,502	7,501	38,542.3	37,022.1	701.1	37,875	36,407	22,502	7,501	6,544.4	3,209.7	2,535.2	44,466	109,695
7,500.B	22,502	7,501	38,528.6	37,024.5	700.7	37,889	36,369	22,502	7,501	6,329.9	3,016.2	2,517.2	44,485	109,677
7,500.C	22,502	7,501	38,532.5	37,004.0	701.5	37,890	36,374	22,502	7,501	6,341.6	2,964.6	2,562.0	44,445	109,532
7,500.D	22,502	7,501	36,466.7	34,998.8	671.7	37,907	36,311	22,502	7,501	6,172.0	2,822.3	2,538.8	44,415	109,698
7,500.E	22,502	7,501	37,540.8	36,076.9	671.3	37,893	36,369	22,505	7,501	6,345.6	2,997.1	2,538.9	44,451	109,541
Average	22,502	7,501	37,922.2	36,425.3	689.3	37,890.8	36,366.0	22,502.6	7,501	6,346.7	3,002.0	2,538.4	44,452.4	109,628.6

Notes. Statistics reported: nNDP, number of points output by the algorithm; nIPF, number of different integer part solutions; TT, total time to discover the NDF; IPT, total time spent in IP solves; LPT, total time spent in liner program (LP) solves; nIP, total number of mixed integer linear programs (IPs) solved; nLex, total number of lexicographic IPs; nMin, total number of single-objective IPs solved to find the nondominated point (NDP) that dominates an open endpoint; nScal, total number of scalarized IPs; nGood: total number of IPs with the no-good constraint; nLP, total number of LPs solved; nRec, total number of boxes processed; nSIS, total number of boxes processed where the corner points are generated by the same integer solution; nZL, number of boxes with z^* on the horizontal split line. Times are reported in seconds.

for class C320 instances. The main contribution, however (and the authors' primary objective), was demonstrating that more accurate NDF approximations can be achieved (compared with the TSA).

The results for SaR with 10 subregions (Soylu 2018) show substantial improvement over the TSA. On average, SaR produced the NDF 32% faster than the TSA for C160 instances and 7% faster than the TSA for C320 instances, where SaR was faster in 9 of 10 instances, by 37% on average, but struggled with one problematic instance. We note that SaR is especially well suited for the historical instances because the NDFs consist of a relatively small number of continuous sections generated by relatively few slices, each having many (small) line segments.

Acknowledgments
The authors thank Banu Soylu for her generosity in sharing her εTCM code with them and for her insightful responses to their questions.

Endnotes

[1] Note that our definition of a slice differs from the original definition by Belotti et al. (2013), where it is defined as the *feasible set* for the slice problem as opposed to the resulting NDF.

[2] The strict inequality used is a convenient shorthand. It is meant to be interpreted and implemented as $z_1(x) \le z_1^* - \epsilon$ for some $\epsilon > 0$.

[3] An alternative method for solving the slice problem is parametric simplex (Ehrgott 2005).

[4] The concept of an ϵ-frontier is not new; however, as opposed to the definitions in papers such as Papadimitriou and Yannakakis (2000) and Vassilvitskii and Yannakakis (2005), which rely on the notion of relative error, our definition uses the notion of absolute error.

[5] The generated instances are publicly available at https://github.com/t-perini/BOMIPresearch.

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

[7] For the C160 instances, the total runtime of the BLM was 7% longer, on average, than the published results for the TSA, and for the C320 instances, the BLM's runtime is 85% longer.

References

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

Belotti P, Soylu B, Wiecek MM (2013) A branch-and-bound algorithm for biobjective mixed-integer programs. *Optimization Online.* Accessed April 17, 2019, http://www.optimization-online.org/DB_FILE/2013/01/3719.pdf.

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

Boland N, Charkhgard H, Savelsbergh M (2015b) A criterion space search algorithm for biobjective mixed integer programming:

The triangle splitting method. *INFORMS J. Comput.* 27(4): 597–618.

Coello CA (2000) An updated survey of GA-based multiobjective optimization techniques. *ACM Comput. Surveys* 32(2):109–143.

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

Deb K (2001) *Multi-Objective Optimization Using Evolutionary Algorithms*, vol. 16 (John Wiley & Sons, New York).

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

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

Ehrgott M, Gandibleux X, eds. (2002) *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys*, International Series in Operations Research and Management Science, vol. 52 (Kluwer Academic Publishers, Boston).

Farahani RZ, SteadieSeifi M, Asgari N (2010) Multiple criteria facility location problems: A survey. *Appl. Math. Model.* 34(7):1689–1709.

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

Lei D (2009) Multi-objective production scheduling: A survey. *Internat. J. Adv. Manufacturing Tech.* 43(9–10):926–938.

Malczewski J (2006) GIS-based multicriteria decision analysis: A survey of the literature. *Internat. J. Geographical Inform. Sci.* 20(7): 703–726.

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

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

Papadimitriou CH, Yannakakis M (2000) The complexity of tradeoffs, and optimal access of web sources. *Proc. 41st Annual Sympos. Foundations Comput. Sci.* (IEEE Computer Society, Piscataway, NJ), 86–92.

Rais A, Viana A (2011) Operations research in healthcare: A survey. *Internat. Trans. Oper. Res.* 18(1):1–31.

Soylu B (2018) The search-and-remove algorithm for biobjective mixed-integer linear programming problems. *Eur. J. Oper. Res.* 268(1):281–299.

Soylu B, Yıldız GB (2016) An exact algorithm for biobjective mixed integer linear programming problems. *Comput. Oper. Res.* 72(Suppl C): 204–213.

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

Vassilvitskii S, Yannakakis M (2005) Efficiently computing succinct trade-off curves. *Theoret. Comput. Sci.* 348(2–3):334–356.

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

White D (1990) A bibliography on the applications of mathematical programming multiple-objective methods. *J. Oper. Res. Soc.* 41(8):669–691.

Zhou A, Qu BY, Li H, Zhao SZ, Suganthan PN, Zhang Q (2011) Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm Evolutionary Comput.* 1(1):32–49.