Branch-and-bound for biobjective mixed-integer linear programming

Submitted to *INFORMS Journal on Computing*
manuscript (Please, provide the manuscript number!)

Branch-and-bound for biobjective mixed-integer linear programming

Nathan Adelgren*
Department of Mathematics and Computer Science, Edinboro University, PA, USA. nadelgren@edinboro.edu,

Akshay Gupte†
School of Mathematics, University of Edinburgh, Edinburgh, UK. akshay.gupte@ed.ac.uk,

We present a generic branch-and-bound algorithm for finding all the Pareto solutions of a biobjective mixed-integer linear program. The main contribution is new algorithms for obtaining dual bounds at a node, for checking node fathoming, presolve and duality gap measurement. Our branch-and-bound is a decision space search method since the branching is performed on the decision variables, akin to single objective problems. The various algorithms are implemented using a data structure for storing Pareto sets. Computational experiments are carried out on literature instances for empirical analysis of our method. We also perform comparisons against an objective space search algorithm from literature, which show that our branch-and-bound is able to compute the entire Pareto set in significantly lesser time.

Key words: Branch-and-bound; Mixed-integer programming; Multiobjective optimization; Pareto optima; Fathoming rules

History: Submitted November 2019; Revised August 2020

1. Introduction

We present a branch-and-bound (BB) algorithm that computes the Pareto set of a biobjective mixed-integer linear program (BOMILP), formulated as

$$\min_{x} \left\{ \begin{array}{l} f_1(x) := {c^1}^\top x \\ f_2(x) := {c^2}^\top x \end{array} \right\} \quad \text{s.t.} \quad x \in X_I := \left\{ x \in \mathbb{Z}^n_+ \times \mathbb{R}^p_+ \colon Ax \leq b,\, l_i \leq x_i \leq u_i \;\forall i \right\}. \tag{1}$$

The only assumption we make on the above model is a mild and standard one: that $X_I \neq \emptyset$ and $-\infty < l_i < u_i < +\infty$ for all i, in order to have a bounded feasible problem.

* Part of this research was carried out when the first author was a PhD student at Clemson University, USA

† The author was supported by ONR grant N00014-16-1-2725 when he was at Clemson University, USA

BOMILPs belong to the general class of multiobjective optimization [Ehr05] and are an extension of the single objective mixed-integer linear program (MILP) that has been studied for decades. A multiobjective problem is considered solved when the entire set of so-called Pareto optimal solutions has been discovered. A common approach to find these Pareto points has been to scalarize the vector objective (cf. [Ehr06; BKR17]) either by aggregating all objectives into one or by moving all but one objective to the constraints, but doing so does not generate all the Pareto points and supplies a very small part of the optimality information that can otherwise be supplied by the original multiobjective problem. Indeed, it is easy to construct examples of biobjective MILPs where many Pareto solutions are located in the interior of the convex hull of the feasible set, a phenomenon that is impossible with optimal solutions of MILPs. The set of Pareto solutions of a mixed-integer multiobjective problem with a bounded feasible region is equal to the union of the set of Pareto solutions from each slice problem. Here the union is taken over the set of integer feasible values and a slice problem is a continuous multiobjective program obtained by fixing the integer variables to some feasible values. In general, there are exponentially many Pareto solutions. Enumeration of the Pareto set for a pure integer problem has received considerable attention, including iterative approaches [ÖK10; LK13] and lower and upper bounds on the number of Pareto solutions [BJV13; SVS13] under certain assumptions. Algorithms using rational generating functions to enumerate all the Pareto optima in polynomial-time for fixed parameters (either size of decision space or number of objectives) were given in [DHK09; BP12]. There also have been many efforts at finding good approximations of the Pareto set [Say00; Say03; RW05; Gra+14; BJV15].

1.1. Background on existing methods

Algorithms for exact solution of multiobjective mixed-integer problems (MOMILPs) can be broadly classified into three categories depending on the underlying techniques they use: (i) those based on scalarization methods that transform the MOMILP into a MILP with a modified objective or with new constraints, (ii) branch-and-bound algorithms which are decision space search since they divide the feasible region X_I by branching on variables (in a manner similar to solving MILPs), and (iii) those based on objective/criterion space search methods that solve MILPs or multiobjective LPs over subsets of the feasible objective space $f(X_I) := \{(f_1(x), f_2(x)) \colon x \in X_I\}$. Multiobjective pure integer problems have been extensively studied in literature and many scalarization methods have been developed,

either specifically for biobjective problems [RSW06] or the fully general multiobjective case [ER08; PGE10; MF13]. Specific classes of biobjective combinatorial problems also have algorithms for solving them [Vis+98; RW07; SS08; BGP09; LLS14]. Earliest branch-and-bound methods for multiobjective pure integer programs can be found in [KH82; KY83], but since then more sophisticated algorithms have been developed [JLS12; GNE19; PT19]. Recent work of Boland et al. has focused on objective space search methods for biobjective [BCS15a] or triobjective [BCS16a; BCS16b] pure integer programs.

Algorithms specialized for the pure integer case do not extend to the mixed-integer case primarily because of the way they certify Pareto optimality. The Pareto set of a mixed-integer problem is a finite union of graphs of piecewise linear functions, whereas that for a pure integer problem is a finite set of points, and hence Pareto computation and certification of Pareto optimality of a given subset is far more complicated in the former case. In fact, mixed-integer problems can benefit immensely from sophisticated data structures for storing Pareto sets, as shown recently by Adelgren et al. [ABG18]. Barring the objective space search method of [BCS15b], most of the exact algorithms for MOMILP have been based on branch-and-bound (BB); see the reviews [PG17] and [GNE19, Table 1]. Most of these BB algorithms are designed specifically for problems where all the integer variables are binary, either for biobjective [Vin+13; SAD14] or the general multiobjective case [MD05]. Correct node fathoming rules are necessary to guarantee correctness of a BB algorithm. Belotti et al. [BSW16] have proposed sophisticated algorithms, based on solving LPs, for node fathoming rules and checking Pareto optimality, and report some limited preliminary computational results. In principle, this leads to a BB algorithm for BOMILP with general integer variables, however, such an algorithm based on sophisticated node fathoming rules has neither been fully implemented nor extensively tested.

1.2. Summary of our work

Our exact algorithm for general BOMILP is based on the BB method. Although there is certainly merit in studying and developing objective space search methods for solving BOMILP, our choice is motivated by the recognition that there is still much work that can be done to exploit the structure of Pareto points in biobjective problems to improve BB techniques for BOMILP. That is indeed the main contribution of this paper — an exhaustive computational study of ideas that specifically address the biobjective nature of problem (1). Besides the fact that BB operates mainly in the x-space and objective space

search, as the name suggests, operates solely in the f-space, another point of distinction between the two is that the MILPs we consider at each node of the BB tree do not have to be solved to optimality whereas the correctness of the latter depends on MILPs being solved to optimality. Of course, it is to be expected that solving MILPs for a longer time will lead to better convergence results for our BB. Implementing our BB through the callback interface of a MILP solver allows us to utilize the huge computational progress made in different components of BB for MILP (cf. [AW13; Mor+16]).

The main components of any BB for MILP include presolve, preprocessing, primal heuristics, dual bounding via cutting planes, node processing, and branching. We present new algorithms to adapt and extend each of these components to the biobjective case. We begin with presolve; since primal presolve techniques work solely on the feasible region, their implementations in state-of-the-art MILP solvers can be directly used for a BOMILP. However, dual presolve utilizes information from the objective function and hence cannot be used directly for a BOMILP. We are the first to discuss (§3.1) and implement an extension of a variety of dual presolve techniques to the multiobjective setting. Additionally, we show that using one of the primal presolve techniques — probing on integer variables (§3.3), alongside branching reduces the overall computational time. Two different preprocessing algorithms (§3.2) are proposed for generating good primal bounds. Our main body of work is in developing new node processing techniques (§4) for BOMILP. The node processing component takes increased importance for BOMILP since bound sets for a multiobjective problem are much more complicated than those for a single objective problem (cf. §2.2), meaning that generation of valid dual bounds and fathoming of a node is not as straightforward as that for MILPs. At each node, we describe procedures to generate valid dual bounds while accounting for the challenges of biobjective problems and strengthen these bounds through the use of locally valid cutting planes and the solution of single objective MILPs. Our bounds are tighter than what has previously been proposed. To guarantee correctness of our BB, we develop new fathoming rules and delineate their difference to the recent work of [BSW16] in §4.3. A branching scheme is presented in §5.1 and a method for exploiting distances between Pareto points in the objective space is discussed in §5.2. Finally, our BB also incorporates an early termination feature that allows it to terminate after a prescribed gap has been attained. In the MILP case, gap computation is trivial to implement because primal and dual bounds for MILPs are scalars. However

for BOMILPs, since these bounds are subsets of $\mathbb{R}^2$ as explained in §2.2, computation of optimality gap requires the use of error measures that are nontrivial to compute. To aid quicker computation, we propose in §5.3 an approximated version of the Hausdorff metric and computationally compare it to the hypervolume gap measure from literature.

An extensive computational analysis is carried out in §6 on literature instances. The first of these experiments evaluates our three dual presolve techniques and the results show that duality fixing is the most useful of the three for reducing CPU time. In our second experiment, we demonstrate that preprocessing methods utilizing ε-constraint scalarization techniques typically yield better primal bounds at the start of BB than weighted sum scalarization techniques. Next, we evaluate the performance of various procedures, such as probing, objective-space fathoming, a variety of cut generation techniques, and some minor improvements to our proposed fathoming rules, that we propose in this paper for improving the overall performance of BB. These tests indicated that probing prior to each branching decision and objective space fathoming are very useful for decreasing the total solution time. The local cuts that we added were not as useful. Finally, we compared the performance of our BB with that of the triangle splitting method [BCS15b], which we recall is an objective space search method, and observe that our BB uses less CPU time to compute the complete Pareto sets of the test instances.

We conclude this paper with a few remarks in §7. We observe that a majority of the algorithms proposed in this paper can be extended naturally to the multiobjective case. The main challenge in developing a fully implementable and efficient BB algorithm for multiobjective MILP is in carrying out the bound domination step. We present some directions for future research on this topic.

2. Preliminaries

2.1. Definitions and Notation

The idea of optimality for single objective optimization is replaced with the idea of *efficiency* in multiobjective problems. Consider BOMILP (1). For any two points $y, y' \in \mathbb{R}^2$, it is said that y *dominates* y' if $y \leq y'$, or equivalently $y' \in y + \mathbb{R}^2_{\geq 0}$. We express this relationship as $y \succ y'$. Denoting $\boldsymbol{f}(x) := (f_1(x), f_2(x))$, which is a vector in $\mathbb{R}^2$, a point $x \in X_I$ is said to be *efficient* if there is no $x' \in X_I$ such that $\boldsymbol{f}(x') \succ \boldsymbol{f}(x)$. A point in $\mathbb{R}^2$ is Pareto optimal (also called nondominated) if it is the $\boldsymbol{f}$-image of some efficient solution in X_I. Denote the sets of efficient solutions and Pareto optimal solutions, respectively, by

$$X_E := \{x \in X_I \colon x \text{ is efficient}\}, \qquad Y_N = \boldsymbol{f}(X_E) := \{\boldsymbol{f}(x) \colon x \in X_E\}.$$

The nondominated subset of any $S \subset \mathbb{R}^2$ is defined as

$$\mathcal{ND}(S) := \{y \in S : \nexists\, y' \in S \text{ s.t. } y' \succ y\}.$$

Therefore, if we let $Y_I := \{\boldsymbol{f}(x) : x \in X_I\}$, we have that $Y_N = \mathcal{ND}(Y_I)$.

For $k = 1, 2$, let $f_k^* := \min\{f_k(x) : x \in X_I\}$ be the optimal value of objective k for the single objective problem. Denote

$$Y_I^k := \left\{ y \in \mathbb{R}^2 : y_i = f_i^* \;\; i \neq k,\, y_k = \min_{x \in X_I} \{f_k(x) : f_i(x) = f_i^* \;\; i \neq k\} \right\} \quad k = 1, 2.$$

We have $Y_I^k \subset Y_N$. For each of X_I, Y_I, and Y_I^k, dropping the I subscript indicates the continuous relaxation of the set. Also, if we add a subscript s, then it means that the set is associated with node s of the BB tree. We use $\mathcal{OS}$ to denote the *objective space*, i.e., the smallest rectangle in $\mathbb{R}^2$ that contains Y. Given $S \subseteq \mathcal{OS} \subseteq \mathbb{R}^2$, the *ideal point* of S, denoted S^{ideal}, is the point $y \in \mathbb{R}^2$ with $y_k = \min_{y \in S}\{y_k\}$ for $k = 1, 2$.

We assume background in branch-and-cut algorithms for single objective problems (cf. [Mar01]). One of the key differences and challenging aspects of BOMILP versus MILP is the concept of primal and dual bound sets, which we explain next.

2.2. Bound sets for BOMILP

Similar to the single objective case, correct fathoming rules are essential for any BB algorithm to solve BOMILP to Pareto optimality. Primal and dual bounds in a single objective BB are scalars, making it easy to compare them and fathom a node by bound dominance. In biobjective BB, these bounds are subsets of $\mathbb{R}^2$. Bound sets were first discussed by Ehrgott and Gandibleux [EG07]. The manner in which these bound sets are generated within a BB is conceptually similar to the single objective case and we explain this next. Note that our forthcoming explanation trivially extends to the multiobjective case.

Suppose that we are currently at node s of the BB tree. The primal bound sets are constructed from the set of integer feasible solutions, denoted by $T_s \subset \mathbb{Z}^n$, found so far by the BB. For every $\tilde{x} \in T_s$, the BOLP obtained by fixing $x_i = \tilde{x}_i$ for $i = 1, \ldots, n$ in BOMILP (1) is called the *slice problem*. The Pareto curve for this slice problem is $\mathcal{ND}(\boldsymbol{f}(X(\tilde{x})))$, where $X(\tilde{x})$ denotes the feasible set of the slice problem, and this curve is convex (because it is minimization) piecewise linear. Then $\mathcal{N}_s := \mathcal{ND}(\cup_{\tilde{x} \in T_s} \mathcal{ND}(\boldsymbol{f}(X(\tilde{x}))))$ is the globally valid primal bound calculated at node s. For the dual bound set, we consider BOLPs

obtained by relaxing integrality on variables. Since X_s denotes the relaxed feasible set at node s and $Y_s = \boldsymbol{f}(X_s)$, the local dual bound is $\mathcal{L}_s := \mathcal{ND}(Y_s)$ and is convex piecewise linear. The global dual bound $\mathcal{L}_s^{global}$ is obtained by considering the local dual bounds for all the open nodes in the BB tree, i.e., $\mathcal{L}_s^{global} = \mathcal{ND}(\cup_{s' \in \Omega_s} \mathcal{L}_{s'})$ where Ω_s is the set of unexplored nodes so far, and this bound is a union of convex piecewise linear curves.

For multiobjective BB, node s is allowed to be fathomed by bound dominance if and only if $\mathcal{L}_s$ is dominated by $\mathcal{N}_s$, i.e., for every $y' \in \mathcal{L}_s$ there exists a $y \in \mathcal{N}_s$ such that $y \succ y'$. Equivalently, due to translation invariance of $\succ$, we have that node s can be fathomed by bound dominance if and only if $\mathcal{L}_s + \mathbb{R}^2_{\geq 0} \subset \mathcal{N}_s + \mathbb{R}^2_{\geq 0}$. For this reason, henceforth for convenience, we consider our local dual bound to be $\mathcal{L}_s = \mathcal{ND}(Y_s) + \mathbb{R}^2_{\geq 0}$ and the current primal bound to be $\mathcal{U}_s := \mathcal{N}_s + \mathbb{R}^2_{\geq 0}$. Thus the dual bound set is a polyhedron whereas the primal bound is a finite union of polyhedra. Although this deviates from the traditional view of bound sets, which defines them in the previous paragraph in terms of the boundary of these polyhedra, it is clear that there is a one-to-one correspondence between fathoming rules for the two alternate representations of bound sets.

Figure 1 illustrates the concept of bound sets. Here, s_2 can be fathomed because $\mathcal{L}_{s_2} \subset \mathcal{U}_s$ but we cannot say anything about fathoming node s_1 since $\mathcal{L}_{s_1} \not\subseteq \mathcal{U}_s$. As can be imagined from Figure 1, fathoming is even more crucial and computationally expensive for BOMILPs since it involves checking inclusion and intersection of polyhedral sets as opposed to comparing scalar values in the MILP case. Thus, the majority of the computational effort in multiobjective BB is spent processing a node s of the BB tree, in particular checking various fathoming rules.

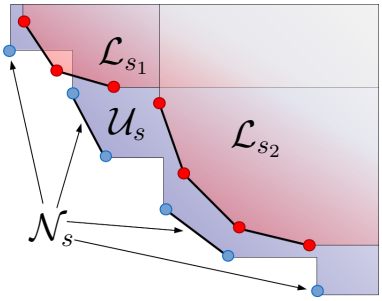


Figure 1 **Primal ($\mathcal{U}$) and dual ($\mathcal{L}$) bound sets for BOMILP**

3. Presolve and Preprocessing

Examining the structure of an instance of single objective MILP prior to solving it, and utilizing information found during this examination to simplify the structure of the instance often has had a significant impact on the time and effort needed to solve that instance. It has also been shown that knowledge of feasible solutions for an instance of MILP can have a significant impact on solution time. Hence, it seems natural as a first step to extend the techniques used in these procedures to the biobjective case. For the discussion that follows we distinguish the idea of simplifying an instance of BOMILP based on its problem structure from the idea of determining a set of initial integer feasible solutions. We refer to the first as *dual presolve* and the latter as *preprocessing*.

3.1. Dual Presolve

Presolve for MILP uses both primal and dual information. The primal information of a BOMILP instance is no different than its single objective counterpart and thus primal presolve techniques can be applied directly to it. However, due to the presence of an additional objective, one must take care while utilizing dual information for a biobjective problem. We extend a few single objective dual presolve techniques to BOMILP (their extension to three or more objectives is immediate and omitted here).

First, we extend duality fixing (cf. [Mar01]). Let a_{rj} denote the element of matrix A in row r and column j and c_j^k be the j^{th} entry of k^{th} objective.

PROPOSITION 1 **(Duality fixing)**. *Suppose there exists a j with $c_j^k \geq 0$ and $a_{ij} \geq 0$ for all k, i. Then $X_E \subseteq \{x\colon x_j = l_j\}$. Similarly, if there exists a j with $c_j^k \leq 0$ and $a_{ij} \leq 0$ for all k, i, then $X_E \subseteq \{x\colon x_j = u_j\}$.*

Proof. It is well known (cf. [Ehr05, Theorem 4.5]) that x^* is efficient for a MOMILP if and only if there exists ε such that x^* is optimal to the problem:

$$\min_x \{f_1(x)\colon x \in X_I,\ f_k(x) \leq \varepsilon_k \text{ for all } k \neq 1\} \tag{2}$$

Hence, every efficient solution to the given BOMILP can be obtained by solving (2) for some ε. If the stated assumptions hold, then single objective duality fixing can be applied to (2). This shows that every efficient solution to the given BOMILP can be obtained by solving the modified version of (2) in which variable fixing has been performed. □

Next, we extend the exploitation of singleton and dominating columns [Gam+15].

PROPOSITION 2 **(Singleton Columns)**. *For every row r in the system $Ax \leq b$, define* $J(r) := \{j\colon a_{rj} > 0, c_j^k < 0\ \forall k, a_{ij} = 0\ \forall i \neq r\}$ *and*

$$U_r := \sum_{j \in J(r)} a_{rj} l_j + \sum_{j \notin J(r), a_{rj} > 0} a_{rj} u_j + \sum_{j \notin J(r), a_{rj} < 0} a_{rj} l_j.$$

Suppose there exists some $s \in J(r)$ such that

$$\frac{c_s^k}{a_{rs}} \leq \min\left\{ \frac{c_t^k}{a_{rt}} \colon t \in J(r), t \neq s \right\}.$$

If $a_{rs}(u_s - l_s) \leq b_r - U_r$, then $X_E \subseteq \{x\colon x_s = u_s\}$.

Proof. Let x be an efficient solution with $x_s < u_s$. If $x_j = l_j$ for all $j \in J(r) \setminus \{s\}$, then a new solution x' constructed from x by setting x'_s to u_s is feasible because

$$\sum_j a_{rj} x'_j = \sum_{j \neq s} a_{rj} x'_j + a_{rs} u_s \leq U_r + a_{rs}(u_s - l_s) \leq b_r.$$

Additionally, the value of every objective function improves because $c_s^k < 0$ for all k. This contradicts our assumption of x being efficient. Hence, there exists a $j \in J(r) \setminus \{s\}$ with $x_j > l_j$. In this case we can construct a new solution x^* from x by decreasing the value of x_j to x'_j while at the same time increasing the value of x_s so that $A_{r\bullet} x^* = A_{r\bullet} x$. In particular, $a_{rs}(x_s^* - x_s) = a_{rj}(x_j - x_j^*)$ holds. The change of objective k can be estimated by

$$\begin{aligned}
c_s^k x_s^* + c_j^k x_j^* &= c_s^k x_s + c_j^k x_j + c_s^k (x_s^* - x_s) - c_j^k (x_j - x_j^*) \\
&= c_s^k x_s + c_j^k x_j + c_s^k \frac{a_{rs}}{a_{rs}} (x_s^* - x_s) - c_j^k \frac{a_{rj}}{a_{rj}} (x_j - x_j^*) \\
&\leq c_s^k x_s + c_j^k x_j + c_s^k \frac{a_{rs}}{a_{rs}} (x_s^* - x_s) - c_s^k \frac{a_{rj}}{a_{rs}} (x_j - x_j^*) \\
&= c_s^k x_s + c_j^k x_j + \frac{c_s^k}{a_{rs}} \left(a_{rs}(x_s^* - x_s) - a_{rj}(x_j - x_j^*) \right) \\
&= c_s^k x_s + c_j^k x_j.
\end{aligned}$$

If $x_s^* = u_s$, the result of the proposition holds. Otherwise, $x_j^* = l_j$ holds. Applying this argument iteratively results in an optimal solution with $x_s^* = u_s$ or $x_j^* = j$ for all $j \in J(r) \setminus \{s\}$. But as shown before, the latter case contradicts the efficiency of x^*. □

A similar procedure can be followed for $a_{rj} < 0$, $c_j^k > 0$ for all k, thereby fixing $x_s = l_s$.

Given two variables x_i and x_j, either both integer or both continuous, we say that x_j *dominates* x_i if (i) $c_j^k \leq c_i^k$ for all k, and (ii) $a_{rj} \leq a_{ri}$ for every r. [1]

[1] This variable domination has no relationship with the idea of domination between bound sets

PROPOSITION 3 **(Dominating columns)**. *If x_j dominates x_i,*

$$Y_N = \{\boldsymbol{f}(x)\colon x \in X_E,\, x_i = l_i \text{ or } x_j = u_j\} \subseteq \{\boldsymbol{f}(x)\colon x \in X_I,\, x_i = l_i \text{ or } x_j = u_j\}.$$

Proof. The $\subseteq$-inclusion is obvious from $X_E \subseteq X_I$, and so we have to argue the equality. We will need the following claim, which can be argued easily and is also an extension of [Gam+15, Lemma 1]: for any $x \in X_I$ with a pair of indices (i,j) such that $x_j < u_j$, $x_i > l_i$, and x_j dominates x_i, the point x^α constructed for arbitrary $0 < \alpha \leq \min\{x_i - l_i, u_j - x_j\}$ as follows,

$$x_i^\alpha = x_i - \alpha, \qquad x_j^\alpha = x_j + \alpha, \qquad x_t^\alpha = x_t, \quad t \neq i,j, \tag{3}$$

satisfies $x^\alpha \in X_I$ and $f_k(x^\alpha) \leq f_k(x)$ for all k.

Since $Y_N = \boldsymbol{f}(X_E)$ by definition, the $\supseteq$-inclusion is obvious. Now suppose for sake of contradiction that the $\subseteq$-inclusion is not true. Then there exists some $y \in Y_N$ for which

$$\boldsymbol{f}^{-1}(y) \bigcap (\{x\colon x_i = l_i\} \cup \{x\colon x_j = u_j\}) = \emptyset. \tag{4}$$

Take any $x \in \boldsymbol{f}^{-1}(y)$, this point has $x_j < u_j$ and $x_i > l_i$. Consider the feasible solution x^α, for $\alpha = \min\{x_i - l_i, u_j - x_j\}$, constructed as in equation (3). By definition of α, we have $x_j^\alpha = u_j$ or $x_i^\alpha = l_i$, and the claim gives us $x^\alpha \in X_I$. We know that $\boldsymbol{f}(x) = y \in Y_N$. Then, $\boldsymbol{f}(x^\alpha) \leq \boldsymbol{f}(x)$ from the above claim implies that $x^\alpha \in \boldsymbol{f}^{-1}(y)$. Hence, we have reached a contradiction to equation (4). □

One may use the disjunction resulting from Proposition 3 to generate valid cutting planes for X_I prior to branching. Additionally, there are also ways to further utilize the structure of dominating columns in order to strengthen variable bounds as described in Gamrath et al. [Gam+15, Theorem 3, Corollary 1 and 2]. These methods for strengthening bounds also extend to the multiobjective case. However, we did not find these methods to be advantageous in our experiments. Thus, since the description of these additional strategies is quite lengthy, we omit them from this work.

3.2. Preprocessing

As in the single objective case, the efficiency of BB can be significantly improved if good-quality primal feasible solutions can be generated prior to the start of BB. This can be accomplished by a heuristic method, such as [Soy15; Lei+16]. We utilize two different preprocessing techniques, both of which solve single objective MILPs subject to a certain

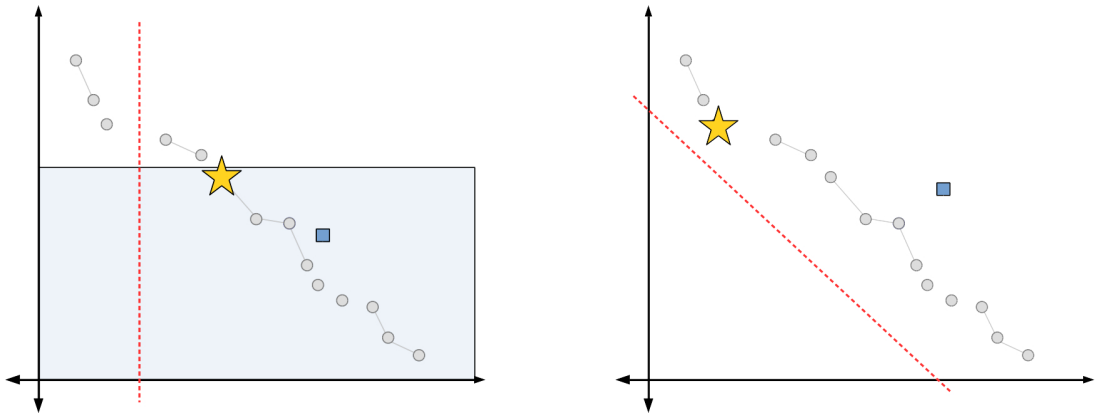
time limitation — the first uses the ε-constraint method, and the second uses the weighted-sum approach. We briefly discuss the benefits and drawbacks of using either the ε-constraint or weighted-sum approaches (see [Ehr05] for background on scalarization methods).

ε-constraint: It is well known that for a BOMILP every $y \in Y_N$ can be obtained using the ε-constraint method. Unfortunately though, when a MILP formulated using the ε-constraint method is not solved to optimality, there are two major drawbacks: (i) each $y \in Y_I$ discovered while processing the MILP must lie within a restricted region of $\mathcal{OS}$, and (ii) the information associated with the best dual bound cannot be utilized.

weighted-sum: The major drawback of the weighted sum method is that when a MILP is formulated using this method, only *supported* Pareto solutions can be found, i.e., those lying on the boundary of the convex hull of Y_N. There are, however, the following two benefits: (i) $y \in Y_I$ discovered during the MILP solve are not restricted to any particular region of $\mathcal{OS}$, and (ii) the best dual bound is valid for all $y \in Y_I$ and can therefore be used to create a cutting plane in $\mathcal{OS}$.

As can be seen, there is a certain level of trade-off present between the ε-constraint method and the weighted sum method. The pros and cons of each technique are illustrated in Figures 2a and 2b. For each of these figures, we have the following: (i) Y_N, which we assume to be unknown, is shown in grey, (ii) the optimal solution, which we assume is not known at termination of the MILP solve, is depicted as a yellow star, (iii) the best known solution at termination is shown as a blue square, and (iv) the level curve associated with the best known dual bound at termination is shown as a dotted red line. Note that for Figure 2a, we assume that ε is defined so that the feasible region is restricted to the light blue box.

We now present Algorithms 1 and 2 in which we describe our proposed ε-constraint and weighted sum based preprocessing procedures. On line 3 of Algorithm 1 we solve the MILP associated with f_λ. Recall that λ is computed so that the level curves of f_λ have the same slope as the line segment joining y_I^1 and y_I^2. On line 5 we then use the solution of this MILP to compute horizontal and vertical step sizes, h_1 and h_2. These step sizes are then used to sequentially increase the values of ε_1 and ε_2 which are used on line 7 to construct new MILPs, using the ε-constraint problem, which may yield new, undiscovered Pareto solutions. On lines 8 and 9 we modify the step sizes h_1 and h_2. If the MILP solved on line 7 yields a new, previously undiscovered Pareto solution, we decrease the step size.



(a) ε-constraint method (b) Weighted sum method

Figure 2 Bound information when a single objective MILP terminates early

Otherwise, we increase it. This allows us the continue searching for additional new solutions in locations of $\mathcal{OS}$ which are near previously discovered solutions, and to cease searching in areas in which new solutions are not being generated. Note that the amount in which the step sizes are increased or decreased depends on the value of the parameter ρ. Also note that each time we solve a MILP, we utilize its solution to update $\mathcal{N}_s$.

In Algorithm 2 we compute several sets of weights which we utilize in the weighted-sum approach to generate Pareto solutions. We initialize the set of weights Λ on line 3 with the weight λ for which the level curves of f_λ have the same slope as the line segment joining y_I^1 and y_I^2. We use σ to represent the number of weights for which MILPs will be solved in a given iteration. We deem an iteration successful if at least a fifth of the solved MILPs reveal previously undiscovered Pareto solutions. We use τ to count the number of unsuccessful iterations. On line 11 we increase the number of weights that will be used in the next iteration by computing the next set of weights so that it contains the midpoint of each pair of adjacent weights in the set Λ', which is the set of previously used weights together with 0 and 1. The process then terminates when the number of unsuccessful iterations exceeds the value of the parameter ρ. As we did with Algorithm 1, we also utilize the solution of each MILP we solve in this procedure to update $\mathcal{N}_s$.

3.3. Probing

After Preprocessing, a probing technique can be used to strengthen the bounds on each integer variable, as stated below.

Algorithm 1 Preprocessing based on the ε-constraint method.

Input: y_I^1, y_I^2 and a nonnegative value for parameter ρ.

Output: An initialized set of Pareto solutions $\mathcal{N}_0 \subseteq Y_N$.

function PREPROCESSINGMETHOD1(y_I^1, y_I^2, ρ)
  Let $\mathcal{N}_0 = \emptyset$.
  Solve the MILP $\min\{f_\lambda(x) : x \in X_I\}$ to obtain $y_I^\lambda \in Y_I$.
  Add a cutting plane to X lying on the level curve of f_λ associated with the best dual solution.
  Set $h_1 = \frac{(y_I^2)_1 - (y_I^\lambda)_1}{60}$, $\varepsilon_1 = (y_I^\lambda)_1 + h_1$, $h_2 = \frac{(y_I^1)_2 - (y_I^\lambda)_2}{60}$ and $\varepsilon_2 = (y_I^\lambda)_2 + h_2$.
  for $k \in \{1, 2\}$ **do**
    while $\varepsilon_k > (y_I^k)_k$ **do**
      Solve the MILP $P_k(\varepsilon_k) := \min\{f_{\{1,2\}\setminus\{k\}}(x) : x \in X_I, f_k(x) \leq \varepsilon_k\}$ to obtain $y^* \in Y_N$.
      if $\mathcal{N}_0 \not\succ y^*$ **then** set $h_k = \frac{h_k}{1+\rho}$.
      else set $h_k = \max(5 - \rho, 1)h_k$.
      for each $x \in X_I$ found while solving $P_k(\varepsilon_k)$ **do**
        Let $N =$ GENERATEDUALBD($s(x)$) and set $\mathcal{N}_0 = \mathcal{ND}(\mathcal{N}_0 \cup N)$.
      Set $\varepsilon_k = \varepsilon_k + h_k$.
  Return $\mathcal{N}_0$.

PROPOSITION 4 **(Probing on x_i).** *Let x_i be an integer variable. Fix $x_i = l_i$, relax integrality on other integer variables and solve the BOLP relaxation to obtain its Pareto set $\mathcal{L}_{l_i}$. If $\mathcal{U}_0 \succ \mathcal{L}_{l_i}$ then $X_E \subseteq \{x : x_i \geq l_i + 1\}$.*

Proof. Recognize that $\mathcal{L}_{l_i}$ dominates every $y \in Y_I$ where $y = \boldsymbol{f}(x)$ with $x_i = l_i$. The desired result follows from $\mathcal{U}_0 \succ \mathcal{L}_{l_i}$. □

This probing procedure can be repeated multiple times for a given integer x_i and then iterated over each additional integer variable x_j. Furthermore, a similar procedure to that of Proposition 4 exists for tightening the upper bound. We point out that there are likely many more tasks that could be performed during Presolve and/or Preprocessing that could further impact the performance of BB. However, our goal here is not to develop extensive procedures for these tasks, but to put together an initial implementation that highlights some of what can be done.

Algorithm 2 Preprocessing based on the weighted-sum method.

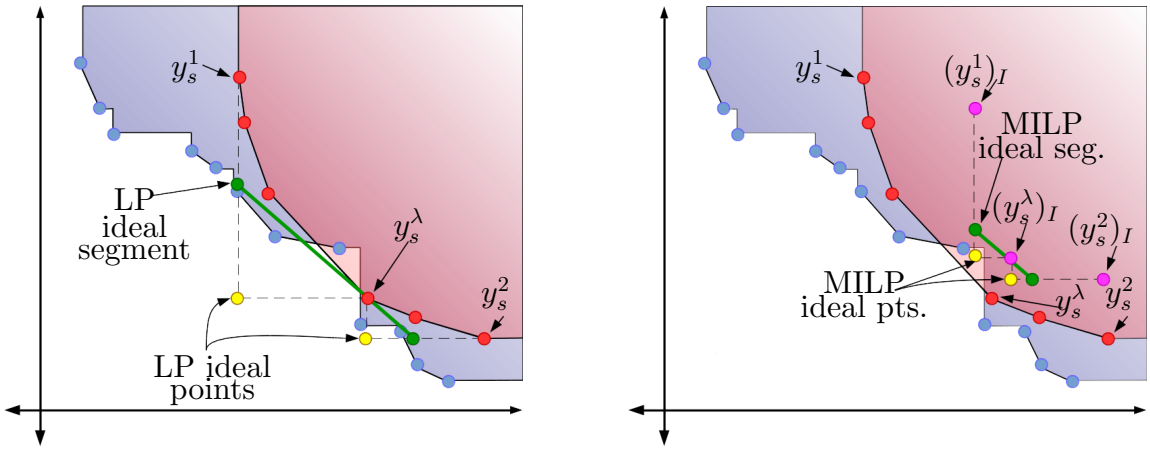
Input: A nonnegative value for parameter ρ.

Output: An initialized set of Pareto solutions $\mathcal{N}_0 \subseteq Y_N$.

function PREPROCESSINGMETHOD2(ρ)
  Let $\mathcal{N}_0 = \emptyset$.
  Set $\Lambda = \{\lambda\}$, $\Lambda' = \{0, 1\}$ and $t = 0$.
  while $t \leq \rho$ **do**
    Set $\tau = 0$ and $\sigma = |\Lambda|$.
    for $\lambda' \in \Lambda$ **do** remove λ' from Λ and add it to Λ'. (Assume Λ' is always sorted in increasing order.)
      Solve the MILP $P(\lambda') := \min\{f_{\lambda'}(x) : x \in X_I\}$ to obtain $y^{\lambda'} \in Y_I$.
      Add a cutting plane to X lying on the level curve of $f_{\lambda'}$ associated with the best dual solution.
      if $\mathcal{N}_0 \not\succ y^{\lambda'}$ **then** set $\tau = \tau + 1$.
      for each $x \in X_I$ found while solving $P(\lambda')$ **do** let $N =$ GENERATEDUALBD($s(x)$) and set $\mathcal{N}_0 = \mathcal{ND}(\mathcal{N}_0 \cup N)$.
    for each adjacent pair $(\lambda_1, \lambda_2) \in \Lambda'$ **do** add $\frac{\lambda_1+\lambda_2}{2}$ to Λ.
    if $\tau < \frac{\sigma}{5}$ **then** set $t = t + 1$.
  Return $\mathcal{N}_0$.

4. Node processing

Processing a node consists of three basic steps: (i) Generate a valid dual bound; (ii) Check a fathoming rule to determine whether or not s can be eliminated from the search tree; (iii) Optionally, if s is not fathomed in (ii), generate a tighter dual bound and repeat (ii). Figure 3 provides a visual example of how one might carry out these three steps. Most of the fathoming rules for biobjective BB are designed to check whether or not $\mathcal{U}_s$ dominates $(Y_s)_I$ by exploiting the transitivity of dominance. First, a set $\mathbb{T}$ is generated such that $\mathbb{T} \succ (Y_s)_I$. Then if $\mathcal{U}_s \succ \mathbb{T}$, $\mathcal{U}_s \succ (Y_s)_I$ and s can be fathomed. Otherwise, a tighter bound on $(Y_s)_I$ is needed. The first bound we use is a set of two ideal points which we obtain by solving three single objective LPs; one for each f_k and an one with a weighted sum objective f_λ in which the weights, denoted λ^s, are given by the normal vector of the line segment H_s passing



(a) LP ideal points and segment

(b) MILP ideal points and segment

Figure 3 Fathoming in biobjective BB

through y_s^1 and y_s^2. We begin with these points because it is straightforward to determine whether or not $\mathcal{U}_s$ dominates a singleton. In Figure 3a these points are labelled "LP ideal points." Notice that they are not dominated. Consider the intersection of $(Y_s)^{ideal} + \mathbb{R}^2_{\geq 0}$ and the line with normal vector λ^s passing through y_s^λ. Recognize that this intersection, which we denote H_s^λ, is also a valid dual bound. In Figure 3a the resulting line segment is labelled "LP ideal segment," but is not dominated. A tighter bound can next be found by explicitly generating $\mathcal{L}_s$. In Figure 3a this is the set indicated by the red points, which is again not dominated. After generating $\mathcal{L}_s$, one cannot hope to find a tighter bound on $(Y_s)_I$ resulting from LP solutions. Instead, one can solve single objective MILPs to generate elements of $(Y_s)_I$ and use these elements to form a valid dual bound. We first generate ideal points in the same way as before, but use single objective MILPs rather than LPs. In Figure 3b these points are labelled "MILP ideal points." Yet again they are not dominated. We can then consider the intersection of $((Y_s)_I)^{ideal} + \mathbb{R}^2_{\geq 0}$ and the line with normal vector λ^s passing through $(y_s^\lambda)_I$, which we denote $\tilde{H}_s^\lambda$. This intersection forms another valid dual bound. In Figure 3b the resulting line segment is labelled "MILP ideal segment" and is dominated. Hence, s can be fathomed in this example.

We now formally outline the fathoming rules employed in this work. Some additional notation will be useful. For $k \in \{1, 2\}$, define

$$\mathcal{P}_s^k := \left(\cup_{i \neq k} y_s^i\right) \cup y_s^\lambda, \tag{5}$$

and let

$$\mathcal{P}_s := (\mathcal{P}_s^1)^{ideal} \cup (\mathcal{P}_s^2)^{ideal}. \tag{6}$$

Additionally, for any $\mathcal{I} \subset \{1, 2, \lambda\}$, define

$$D_s^{\mathcal{I}} := \cup_{k=1}^{2} \left(\left(\mathcal{P}_s^k \setminus \cup_{i \in \mathcal{I}} y_s^i \right) \cup \cup_{i \in \mathcal{I} \setminus \{k\}} (y_s^i)_I \right)^{ideal}. \tag{7}$$

$\mathcal{P}_s$ represents the sets of ideal points obtained from LP solutions, while $D_s^{\mathcal{I}}$ represents a set of ideal points obtained from a mixture of LP and MILP solutions. Our five fathoming rules are given below. Rule 0 expresses the idea of fathoming due to optimality, while the remainder of the rules indicate situations in which s can be fathomed due to bound dominance.

PROPOSITION 5 **(Fathoming Rules)**. *Node s can be fathomed if any of the following holds:*

0. $\mathcal{L}_s \subset (Y_s)_I$,

1a. $\mathcal{U}_s \succ \mathcal{P}_s$,

2a. $\mathcal{U}_s \succ H_s^{\lambda}$,

1b. $\mathcal{U}_s \succ D_s^{\mathcal{I}}$ *for some* $\mathcal{I} \subset \{1, 2, \lambda\}$,

2b. $\mathcal{U}_s \succ \tilde{H}_s^{\lambda}$,

3. $\mathcal{L}_s \subseteq \mathcal{U}_s$.

Proof. Rule 0 is due to integer feasibility of $\mathcal{L}_s$. Rule 1a holds since by construction $\mathcal{P}_s \succ \mathcal{L}_s$, and so $\mathcal{U}_s \succ \mathcal{L}_s$. Rule 2a holds since by construction $\tilde{H}_s^{\lambda} \succ \mathcal{L}_s$, and so $\mathcal{U}_s \succ \mathcal{L}_s$. For Rule 1b, note that by construction, for any $\mathcal{I} \subset \{1, 2, \lambda\}$, $D_s^{\mathcal{I}} \succ (y_s)_I$ for every $(y_s)_I \in (Y_s)_I$ and thus $D_s^{\mathcal{I}}$ is a valid dual bound at node s. For Rule 2b, note that by construction $H_s^{\lambda} \succ (y_s)_I$ for every $(y_s)_I \in (Y_s)_I$ and thus H_s^{λ} is a valid dual bound at node s. Rule 3 is obvious. □

Before we outline the process we use for processing a node s, we briefly discuss another important task that ought to be carried out while processing node s: Updating $\mathcal{N}_s$. We do this in two ways: (i) add each integer-feasible line segment discovered while checking Fathoming Rule 0 to $\mathcal{N}_s$, and (ii) for each discovered $x^* \in X_I$, generate the nondominated subset of

$$\mathcal{Y}(x^*) := \{y = \boldsymbol{f}(x) : x \in X, x_i = x_i^* \text{ for all } i \in \{m+1, \ldots, m+n\}\} \tag{8}$$

and add each defining line segment of this set to $\mathcal{N}_s$. Consider the latter of these strategies. Observe that the feasible set of $\mathcal{Y}(x^*)$ can be interpreted as a leaf node of the BB tree,

which we denote $s(x^*)$. Hence, the $\mathcal{Y}(x^*)+\mathbb{R}^2_{\geq 0} = \mathcal{L}_{s(x^*)}$. This leads to a need for generating the nondominated subset of $\mathcal{L}_s$, i.e. $\mathcal{ND}(\mathcal{L}_s)$. Typical techniques for generating $\mathcal{ND}(\mathcal{L}_s)$ include the multiobjective simplex method and the parametric simplex algorithm (PSA) [Ehr05]. However, the multiobjective simplex method is far more robust than is necessary for biobjective problems. Also, we found in practice that using the PSA often resulted in many basis changes yielding the same extreme point of $\mathcal{L}_s$ in $\mathcal{OS}$. Since much work is done during the PSA to determine the entering and exiting variables, we found that generating $\mathcal{ND}(\mathcal{L}_s)$ using the PSA required a significant amount of computational effort. We decided to use an alternative method for generating $\mathcal{ND}(\mathcal{L}_s)$ which relies on sensitivity analysis. We first solve the single objective LP using objective f_2 to obtain y^2_s. Next we create the LP

$$\mathcal{P}_s(\alpha) := \min\{f_1(x) + \alpha f_2(x) : x \in X_s\} \tag{9}$$

and then carry out the procedure outlined in Algorithm 3.

Algorithm 3 Generate $\mathcal{ND}(\mathcal{L}_s)$

Input: Node s.

Output: A set $\mathcal{B}$ containing all defining line segments of $\mathcal{ND}(\mathcal{L}_s)$.

1: **function** GENERATEDUALBD(s)
2: Set $\mathcal{B} = \emptyset$.
3: Solve the LP $\min\{f_2(x) : x \in X_s\}$ to obtain y^2_s.
4: Solve $\mathcal{P}_s(0)$ to obtain solution x^* and set $y = \boldsymbol{f}(x^*)$.
5: **while** $y \neq y^2_s$ **do**
6: Use sensitivity analysis to obtain an interval $[\alpha', \alpha'']$ such that x^* is optimal to $\mathcal{P}_s(\alpha)$ for all $\alpha \in [\alpha', \alpha'']$.
7: Let α^* be the negative reciprocal of the slope of the line through y and y^2_s.
8: Set $x^* = \arg\min\{\mathcal{P}_s(\alpha'' + \varepsilon)\}$ for sufficiently small $\varepsilon \in (0, \alpha^* - \alpha'']$.
9: **if** $\boldsymbol{f}(x^*) \neq y$ **then**
10: Add the line segment connecting $\boldsymbol{f}(x^*)$ and y to $\mathcal{B}$. Update y to be $\boldsymbol{f}(x^*)$.
11: Return $\mathcal{B}$.

In lines 3 and 4 of Algorithm 3 we compute the south-east and north-west most extreme points of $\mathcal{ND}(\mathcal{L}_s)$, respectively. The while loop beginning on line 5 is then used to sequen-

tially compute adjacent extreme points of $\mathcal{ND}(\mathcal{L}_s)$ in a west to east pattern, until the south-east most extreme point is rediscovered. Each line segment joining a pair of adjacent extreme points of $\mathcal{ND}(\mathcal{L}_s)$ is stored and the set of all computed segments is returned at the end of the procedure. Note that the correctness of the algorithm relies on an appropriately small choice for ε on line 8. As we have discussed, there are other methods which can be used here that do not rely on ε, such as the PSA or the first phase of the two-phase method for solving biobjective combinatorial problems [Ehr05]. We have already discussed the difficulties we encountered with the PSA. The difficulty with the first phase of the two-phase method is that, although it generates the extreme supported Pareto solutions of a BOLP, it does not generate them in order from left to right. Thus, when using a simplex-style solution method for each single objective LP, each iteration can require a significant number of basis changes. Our method generates these extreme points in order from left to right, and as a result, warm-starting each iteration by reusing the basis information from the previous iteration reduces the overall number of required basis changes.

Recognize from Proposition 5 that Fathoming Rules 0 and 3 each impose a condition on $\mathcal{L}_s$ and therefore require knowledge of $\mathcal{ND}(\mathcal{L}_s)$ in order to be employed. We note, however, that for each of these rules it is often unnecessary to generate $\mathcal{ND}(\mathcal{L}_s)$ entirely. In particular, the generation of $\mathcal{ND}(\mathcal{L}_s)$ should cease if: (i) one is checking Fathoming Rule 0 and a defining line segment of $\mathcal{ND}(\mathcal{L}_s)$ is generated that is not integer feasible, or (ii) one is checking Fathoming Rule 3 and a defining line segment of $\mathcal{ND}(\mathcal{L}_s)$ is generated that is not contained in $\mathcal{U}_s$. Hence, the procedures in Algorithm 3 can be modified in order to develop strategies for checking Fathoming Rules 0 and 3. These strategies are outlined in Algorithms 4 and 5, respectively.

Algorithm 4 follows almost the same procedure as Algorithm 3, except it terminates prematurely on line 10 if a line segment is computed that is not integer feasible. Algorithm 5 also follows almost the same procedure as Algorithm 3. However, this procedure terminates prematurely on line 5 or 12 if a point or line segment is computed that is not dominated by $\mathcal{U}_s$. We have now built the tools necessary to present our proposed procedure for processing a node s. We do so in Algorithm 6.

Line 2 of Algorithm 6 is an optional procedure in which we can generate locally valid cutting planes to strengthen the representation of X_s if so desired. We then compute y_s^1 and y_s^2 on line 3. We then check to see if either of these solutions are integer feasible, and

Algorithm 4 Fathoming Rule 0

Input: Node s and solutions y_s^1 and y_s^2.

Output: 1 if node s should be fathomed, 0 otherwise.

1: **function** FR_0(s, y_s^1, y_s^2)
2: y_s^1 is the solution to $\mathcal{P}_s(0)$. Let x^* represent the preimage of y_s^1. Set $y = y_s^1$.
3: **if** $y = y_s^2$ **then** return 1
4: **else**
5: **while** $y \neq y_s^2$ **do**
6: Use sensitivity analysis to obtain an interval $[\alpha', \alpha'']$ such that x^* is optimal to $\mathcal{P}_s(\alpha)$ for all $\alpha \in [\alpha', \alpha'']$.
7: Let α^* be the negative reciprocal of the slope of the line through y and y_s^2.
8: Set $x^* = \arg\min\{\mathcal{P}_s(\alpha'' + \varepsilon)\}$ for sufficiently small $\varepsilon \in (0, \alpha^* - \alpha'']$.
9: **if** $\boldsymbol{f}(x^*) \neq y$ **then**
10: Let $\mathbb{S}$ represent the line segment connecting $\boldsymbol{f}(x^*)$ and y.
11: **if** $\mathbb{S} \not\subset (Y_s)_I$ **then** return 0
12: **else** Update y to be $\boldsymbol{f}(x^*)$.
13: return 1

if they are, we generate the dual bound associated with the integer solution in order to update $\mathcal{N}_s$. Furthermore, if both solutions are integer feasible, we check Fathoming Rule 0 on line 6. On line 7 we compute the value λ_s, the value of the weights on the objectives so that the level curves of f_λ have the same slope as the line segment joining y_s^1 and y_s^2. We then solve the LP associated with f_λ. If the solution is integer feasible, we again update $\mathcal{N}_s$ as before. On line 9 we check whether or not y_s^1, y_s^2 and y_s^λ are dominated by $\mathcal{U}_s$. If they are, we proceed to check Fathoming Rules 1a, 2a, and 3. Otherwise, we solve the MILP associated with f_λ and f_k for each $k \in \{1, 2\}$ such that the ideal point $(\mathcal{P}_s^k)^{ideal}$ is not dominated by $\mathcal{U}_s$. On lines 21 and 22 we utilize the solutions of each MILP to (optionally) add local cuts to X_s and update $\mathcal{N}_s$. Finally, we check Fathoming Rules 1b and 2b.

Two additional tasks are performed while processing each node.

4.1. Objective space fathoming

After processing each node, we perform an additional type of fathoming which we refer to as *objective-space fathoming*. After updating $\mathcal{N}_s$, we impose bounds on f_1 and f_2 which

Algorithm 5 Fathoming Rule 3

Input: Node s and solutions y_s^1 and y_s^2.

Output: 1 if node s should be fathomed, 0 otherwise.

function FR_3(s, y_s^1, y_s^2)
  y_s^1 is the solution to $\mathcal{P}_s(0)$. Let x^* represent the preimage of y_s^1. Set $y = y_s^1$.
  if $y = y_s^2$ **then**
    if $\mathcal{U}_s \succ y$ **then** return 1
    else return 0
  else
    while $y \neq y_s^2$ **do**
      Use sensitivity analysis to obtain an interval $[\alpha', \alpha'']$ such that x^* is optimal to $\mathcal{P}_s(\alpha)$ for all $\alpha \in [\alpha', \alpha'']$.
      Let α^* be the negative reciprocal of the slope of the line through y and y_s^2.
      Set $x^* = \arg\min\{\mathcal{P}_s(\alpha'' + \varepsilon)\}$ for sufficiently small $\varepsilon \in (0, \alpha^* - \alpha'']$.
      if $\boldsymbol{f}(x^*) \neq y$ **then**
        Let $\mathbb{S}$ represent the line segment connecting $\boldsymbol{f}(x^*)$ and y.
        if $\mathcal{U}_s \not\succ \mathbb{S}$ **then** return 0
        else Update y to be $\boldsymbol{f}(x^*)$.
    return 1

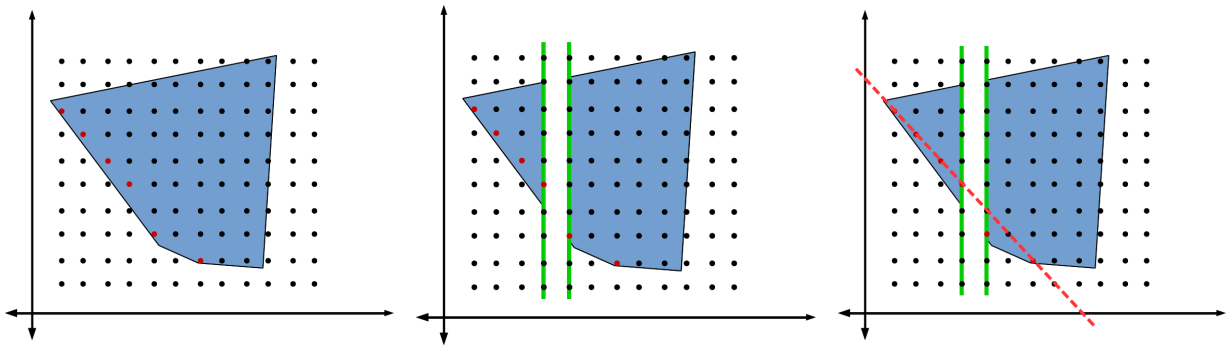
"cut off" portions of $\mathcal{OS}$ in which we have discovered that $\mathcal{U}_s \succ (Y_s)_I$. In certain cases the remaining subset of $\mathcal{OS}$ consists of disjoint regions. When this is the case, we implement objective-space fathoming by branching on f_1 and f_2 bounds which generate the desired disjunctions in $\mathcal{OS}$. In these cases, objective-space fathoming resembles the "Pareto branching" of Stidsen et al. [SAD14]and "objective branching" of Parragh and Tricoire [PT19].

4.2. Bound tightening

In order to increase the likelihood of fathoming, we utilize a few different strategies for tightening the bound $\mathcal{L}_s$. The first strategy we use is the generation of locally valid cutting planes. We do this in two ways: (i) we generate discjuntive cuts based on disjunctions observed in $\mathcal{OS}$ when performing $\mathcal{OS}$ fathoming, and (ii) we convert the BOLP relaxation associated with s to the BOMILP $\min\{f_\lambda(x) : x \in (X_s)_I\}$, allow the MILP solver to process

Algorithm 6 Process node s

function ProcessNode(s)
  Compute valid cutting planes for $(X_s)_I$ and add them to the description of X_s.
  for $k \in \{1,2\}$ **do** Solve $\min\{f_k(x) : x \in X_s\}$ to find optimal solution $\bar{x}^k$ and generate $y_s^k \in Y_s^k$.
    if $y_s^k \in (Y_s)_I$ **then** let $N =$ GenerateDualBd($s(\bar{x}^k)$) and set $\mathcal{N}_s = \mathcal{ND}(\mathcal{N}_s \cup N)$.
  if $y_s^1, y_s^2 \in (Y_s)_I$ **then**
    if FR_0$(s, y_s^1, y_s^2) = 1$ **then** Fathom s, STOP! (Fathoming Rule 0)
  Calculate H_s and λ^s using y_s^1 and y_s^2. Solve $\min\{f_\lambda(x) : x \in X_s\}$ to find optimal solution $\bar{x}^\lambda$ and generate $y_s^\lambda \in Y_s^\lambda$.
  if $y_s^\lambda \in (Y_s)_I$ **then** let $N =$ GenerateDualBd($s(\bar{x}^\lambda)$) and set $\mathcal{N}_s = \mathcal{ND}(\mathcal{N}_s \cup N)$.
  if $\mathcal{U}_s \succ y_s^1$, $\mathcal{U}_s \succ y_s^2$ and $\mathcal{U}_s \succ y_s^\lambda$ **then**
    if $\mathcal{U}_s \succ \mathcal{P}_s$ **then** Fathom s, STOP! (Fathoming Rule 1a)
    else
      Calculate $\tilde{H}_s^\lambda$.
      if $\mathcal{U}_s \succ \tilde{H}_s^\lambda$ **then** Fathom s, STOP! (Fathoming Rule 2a)
      else
        if FR_3$(s, y_s^1, y_s^2) = 1$ **then** Fathom s, STOP! (Fathoming Rule 3)
  else
    Define the set $\mathcal{I} = \emptyset$.
    for $k \in \{1,2\}$ **do**
      if $\mathcal{U}_s \not\succ (\mathcal{P}_s^k)^{ideal}$ **then** add $(\{1,2\} \setminus \{k\}) \cup \{\lambda\}$ to $\mathcal{I}$
    for each $k \in \mathcal{I}$ **do** solve the MILP $\min\{f_k(x) : x \in (X_s)_I\}$ to find optimal solution $\hat{x}^k$ and obtain $(y_s^k)_I \in (Y_s^k)_I$.
      Add a local cut to X_s lying on the level curve of f_k associated with the best dual solution.
      Let $N =$ GenerateDualBd($s(\hat{x}^k)$) and set $\mathcal{N}_s = \mathcal{ND}(\mathcal{N}_s \cup N)$.
    if $\mathcal{U}_s \succ D_s^{\mathcal{I}}$ **then** Fathom s, STOP! (Fathoming Rule 1b)
    else if $\lambda \in \mathcal{I}$ **then**
      Calculate H_s^λ.
      if $\mathcal{U}_s \succ H_s^\lambda$ **then** Fathom s, STOP! (Fathoming Rule 2b)



(a) Example instance of BOMILP (b) After branching (c) Locally valid cut

Figure 4 An example showing the usefulness of locally valid cuts for BOMILP

its root node, and add all cuts generated by this solver as local cuts to s as local cuts. It is widely accepted that for single objective MILPs, locally valid cutting planes are not particularly helpful for improving the performance of BB. However, locally valid cutting planes can have a significantly greater impact on BOMILPs. To see this, observe Figure 4. Assume that Figure 4a displays an instance of BOMILP for which the (f_1, f_2)-space and the X-space are one and the same, i.e., this instance contains only two variables y_1 and y_2, both integer, and $f_1 = y_1$ and $f_2 = y_2$. The constraints of this instance yield the blue polytope, and the integer lattice is indicated by the black dots. The red dots represent the Pareto-optimal solutions. Suppose that branching is performed as shown in Figure 4b. Notice that all Pareto optimal solutions in the left branch can be revealed by a single locally valid cutting plane, as shown by the red dashed line in Figure 4c. Also notice that this could never be accomplished through the use of globally valid cuts.

4.3. Comparison with another BB

We highlight some key differences regarding the node processing step between our BB and that of Belotti et al. [BSW12; BSW16], which is the only other BB method for general BOMILP. There are also differences in the other components of BB, but that is not of concern here.

The two methods differ in the way fathoming rules are implemented. Firstly, we utilize the data structure of Adelgren et al. [ABG18] to store and dynamically update the set $\mathcal{N}_s$ throughout the BB process. In [BSW12; BSW16], fathoming rules are checked at a node s of the BB tree by: (i) using $\mathcal{N}_s$ to generate $\mathcal{U}_s$ by adding a set of local nadir points to $\mathcal{N}_s$, (ii) selecting the subset $\mathcal{R} := \mathcal{U}_s \cap ((Y_s)^{ideal} + \mathbb{R}^2_{\geq 0})$, and (iii) solving auxiliary LPs to determine whether $\mathcal{R}$ and $\mathcal{L}_s$ can be separated by a hyperplane. Node s is then fathomed

if $\mathcal{R} = \emptyset$ or if a separating hyperplane is found. Note that these procedures amount to comparing each element of the primal bound with the dual bound as a whole by solving at most one LP for each element of the primal bound.

In this paper, we utilize the opposite approach to fathoming. Rather than comparing each element of the primal bound with the dual bound as a whole, we compare each element of the dual bound with the primal bound as a whole. Additionally, instead of making these comparisons by solving LPs, we exploit the following guarantee of the data structure of [ABG18]: a point or line segment inserted to the structure is added to the structure if and only if the point or segment is not dominated by the data already stored in the structure. Hence, we implement an extra function IsDominated$(\cdot)$ alongside this data structure which returns 1 if the input is dominated by $\mathcal{N}_s$ and 0 otherwise. We then implement our fathoming rules 1-3 by passing the appropriate sets ($\mathcal{P}_s, H_s^\lambda, D_s^{\mathcal{I}}, \tilde{H}_s^\lambda$ and $\mathcal{L}_s$) to IsDominated. If a 1 is returned for any of these sets, we fathom, otherwise we do not. It is difficult to comment on whether solving LPs or utilizing a function call to a data structure is more efficient for checking fathoming. However, we have found in practice that for a particular node s of the BB tree, the primal bound $\mathcal{U}_s$ typically contains far more points and segments than the dual bound $\mathcal{L}_s$. Thus, comparing each element of the dual bound with the primal bound as a set seems to be a more efficient procedure than doing it the opposite way.

We now discuss the extension of the remaining major aspects of BB to the biobjective setting.

5. Biobjective BB

In this section we discuss the specifics of how the different components of single objective BB — presolve/preprocessing, node processing, and branching, can each be extended to the biobjective setting. We then briefly discuss optional additions to our basic biobjective BB procedure.

5.1. Branching

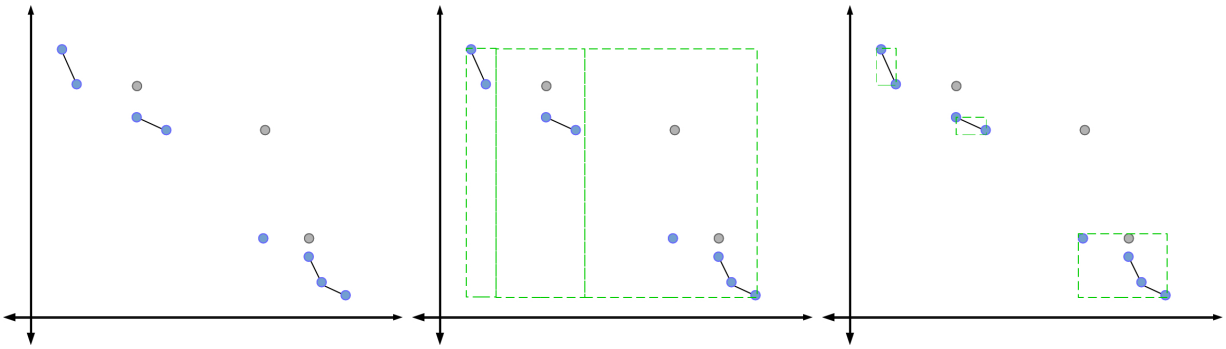
In general, any rule for selecting a branching variable is permissible. However, it should be noted that for BOMILP several $y \in Y$, and consequently several $x \in X$, may be discovered while processing a node s. In fact, our implementation requires solving at least three LPs at each node. Since the variables may take on different values at each solution, it is

possible that an integer variable takes a fractional value at some of these solutions and not at others. Because of this, we use a scoring scheme for branching in which each integer variable is given a score. Of the variables with the highest score, the one with the highest index is selected for branching. The score of x_i is increased if: (i) x_i is fractional at the LP solution associated with objective f^k, $k \in \{1, 2, \lambda^s\}$, (ii) x_i changes value at a pivoting step of Algorithm 4, or (iii) multiple single objective MILPs are solved to optimality at s and x_i takes different values for at least two of the MILP solutions.

After a branching decision has been made we utilize probing, as introduced in Proposition 4, to strengthen bounds on each variable for both of the resulting subproblems. We do this for several reasons: (i) we may find during this process that our branching decision results in an infeasible subproblem, in which case we can discard the infeasible subproblem, enforce that the variable bounds associated with the feasible subproblem be satisfied at any child node of s, and choose a new branching variable; (ii) because much work in biobjective BB is dedicated to fathoming, we want to generate the strongest dual bound possible, which probing helps us to do; (iii) since processing a node in biobjective BB is an expensive operation, we seek to limit the number of nodes explored and probing aids in this endeavor by reducing the number of possible future branching decisions. We found during testing that this probing scheme at each node was extremely powerful, both in reducing the number of nodes processed during BB as well as overall running time. See Table 1 in Section 6 for evidence of this.

5.2. Exploiting gaps in $\mathcal{OS}$

Due to the noncontinuous, nonconvex nature of the Pareto set of a BOMILP, there are occasionally large gaps between Pareto solutions in $\mathcal{OS}$. If this occurs, the likelihood that $\mathcal{L}_s \subseteq \mathcal{U}_s$ is significantly decreased for each node. Hence, this can result in an extreme amount of computational effort which yields no additional Pareto solutions. One way to combat this issue is to observe the solutions obtained during Preprocessing and record locations in $\mathcal{OS}$ where large gaps exist between discovered solutions. One can then split $\mathcal{OS}$ into a series of subregions based on the locations of these gaps and solve single objective MILPs (using objectives f_1 and f_2) within each subregion in order to remove locations containing no Pareto solutions. Afterwards BB can be run in each subregion rather than over the entire $\mathcal{OS}$. To aid in understanding this idea, observe Figure 5. Here Pareto solutions are shown in blue and subregions in $\mathcal{OS}$ are indicated by green dashed lines.



(a) Gaps (b) Slitting $\mathcal{OS}$ (c) Reducing the subregions

Figure 5 Large gaps between solutions in $\mathcal{OS}$

5.3. Measuring Performance

In single objective BB, one can terminate the procedure at any time and obtain a measure of the quality of the best known solution in terms of the gap between this solution and the best known dual bound. We propose a similar scheme for biobjective BB. Let $\mathcal{O}_{s^*}$ represent the set of open nodes after a node s^* has been processed. After processing s^*, the global dual bound is $\mathcal{DB}_{s^*} = \mathcal{ND}(\cup_{s\in\mathcal{O}_{s^*}} \mathcal{L}_s)$. Therefore, if BB is terminated after s^* is processed, the performance of BB can be quantified by measuring the distance between $\mathcal{DB}_{s^*}$ and $\mathcal{U}_{s^*}$. One natural metric to use for measuring this distance is the Hausdorff metric:

$$d_H(\mathcal{DB}_{s^*},\mathcal{U}_{s^*}) := \max\left\{ \sup_{i\in\mathcal{DB}_{s^*}} \inf_{j\in\mathcal{U}_{s^*}} d(i,j),\ \sup_{j\in\mathcal{U}_{s^*}} \inf_{i\in\mathcal{DB}_{s^*}} d(i,j) \right\}.$$

Unfortunately the nonconvex nature of $\mathcal{U}_s$ makes the Hausdorff metric difficult to use since it cannot be computed using a linear program. In our implementation $\mathcal{U}_{s^*}$ is stored as the individual line segments and singletons comprising $\mathcal{N}_{s^*}$ using the data structure of [ABG18]. $\mathcal{DB}_{s^*}$ is computed by generating the points and line segments comprising its nondominated subset, which are also stored using the same data structure. Thus, rather than explicitly computing $d_H(\mathcal{DB}_{s^*},\mathcal{U}_{s^*})$, we instead compute

$$\mathcal{G}_{s^*} := \max\{d_H(\mathcal{DB}_{s^*}, \mathcal{S} + \mathbb{R}^2_{\geq 0}) : \mathcal{S} \in \mathcal{N}_{s^*}\}$$

via pairwise comparison of the points and line segments comprising $\mathcal{DB}_{s^*}$ and $\mathcal{N}_{s^*}$. Clearly, $\mathcal{G}_{s^*}$ is a upper bound on $d_H(\mathcal{DB}_{s^*},\mathcal{U}_{s^*})$. Recognize, though, that $\mathcal{G}_{s^*}$ is an absolute measurement and so it is difficult to use to compare the performance of BB on multiple instances of BOMILP. Thus, in practice we use a percentage calculated as

$$\bar{\mathcal{G}}_{s^*} := 100 \times \frac{|\max\{y^2_1 - y^1_1, y^1_2 - y^2_2\} - \mathcal{G}_{s^*}|}{\max\{y^2_1 - y^1_1, y^1_2 - y^2_2\}}.$$

We refer to this number as the % duality gap.

Another method for measuring the distance between $\mathcal{DB}_{s^*}$ and $\mathcal{U}_{s^*}$ is to compute a so called *hypervolume gap*. Let $hv(\cdot)$ denote the area of subset of $\mathbb{R}^2$. Then the hypervolume gap between $\mathcal{DB}_{s^*}$ and $\mathcal{U}_{s^*}$, as proposed by Zitzler et al. [Zit+03], is

$$\mathcal{HV}_{s^*} := 100 \times \frac{hv((\mathcal{DB}_{s^*} + \mathbb{R}^2_{\geq 0}) \cap \mathcal{OS}) - hv(\mathcal{U}_{s^*} \cap \mathcal{OS})}{hv((\mathcal{DB}_{s^*} + \mathbb{R}^2_{\geq 0}) \cap \mathcal{OS})},$$

A similar measure is used to assess the quality of approximations to the Pareto sets of BOMILP instances in [BCS15b].

Recognize that the Hausdorff and hypervolume gap measurements play significantly different roles. The hypervolume gap provides a measure of the proximity of the dual bound to the primal bound throughout the entirety of $\mathcal{OS}$, while the Hausdorff gap provides a measure of the proximity of the dual and primal bounds in the location at which they are furthest apart. Hence, we can interpret the Hausdorff gap as a worst-case measurement and the hypervolume gap as a sort of average-case measurement. We note that in our initial tests we utilize both the Hausdorff and hypervolume measurements so that our results can be compared with other works, such as [BCS15b], which use the hypervolume gap. However, since the Hausdorff gap provides a worst-case measure and is therefore more robust, we do not use the hypervolume gap measurement in our final set of experiments.

5.4. Our BB algorithm

A pseudocode of our BB procedure is given in Algorithm 7.

6. Computational Analysis

We implemented Algorithm 7 for our BB scheme using the C programming language and the ILOG CPLEX optimization package. This implementation, along with the instances we generated for use in Section 6.6 can be found at `https://github.com/nadelgr/BOMILP_BB`. Boland et al. [BCS15b] graciously shared their code with us and so we were able to compare the performance of our BB with the triangle splitting (TS) method, which we recall is a search method in the objective space. In preliminary tests, we also compared with the BB method of [BSW12]. However, their implementation was incomplete and so the performance of our BB was far superior to theirs. For this reason, we do not include the results of their BB. All testing described in Sections 6.1–6.5 was conducted using a Dell PowerEdge R430 server running Fedora Core 27 and which had a Xeon E5-2640 CPU and 64 GB of

Algorithm 7 BB for BOMILP.

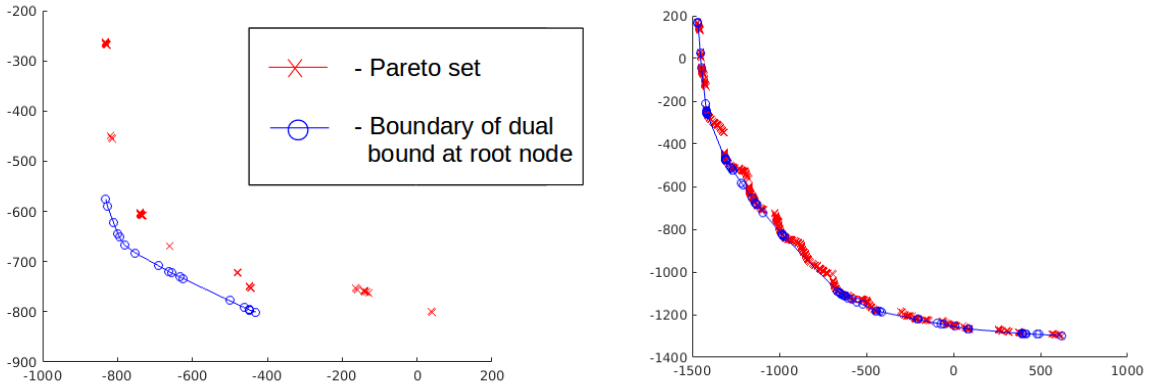
Input: An instance $\mathcal{I}$ of BOMILP.

Output: The Pareto set of instance $\mathcal{I}$.

1: **function** BBSOLVE($\mathcal{I}$)
2: Set $\mathcal{L} = \emptyset$.
3: Use primal presolve, biobjective duality fixing and exploitation of singleton and dominating columns to simplify $\mathcal{I}$.
4: **for** $k \in \{1, 2\}$ **do** solve the MILP $\min\{f_k(x) : x \in X_I\}$ to obtain $y_I^k \in Y_I$.
5: Select $\rho \geq 0$ and run either PREPROCESSINGMETHOD1(y_I^1, y_I^2, ρ) or PREPROCESSINGMETHOD2(y_I^1, y_I^2, ρ) to return $\mathcal{N}_0$.
6: Perform probing to further simplify $\mathcal{I}$.
7: Add the continuous relaxation of $\mathcal{I}$ to $\mathcal{L}$.
8: **while** $\mathcal{L} \neq \emptyset$ **do** select s from $\mathcal{L}$.
9: Run PROCESSNODE(s).
10: **if** s is not fathomed **then** perform $\mathcal{OS}$ fathoming.
11: **if** the nondominated portion of $\mathcal{OS}$ consists of disjoint regions **then** perform Pareto branching. Add the resulting subproblems to $\mathcal{L}$.
12: **else** select the variable with highest score for branching.
13: Perform probing to simplify each of the subproblems resulting from the current branching decision.
14: **if** probing reveals an infeasible subproblem **then** impose the restrictions of the feasible subproblem and select the variable with the next highest score for branching. Repeat Line 13.
15: **else** branch on the selected variable. Add the resulting subproblems to $\mathcal{L}$.
16: Return $\mathcal{N}_{s^*}$, where s^* is the last node for which PROCESSNODE was called.

RAM. For tests described in Section 6.6 we utilized the Extreme Science and Engineering Discovery Environment (XSEDE) Bridges system at the Pittsburgh Supercomputing Center (PSC) through allocation DMS200019. Specifically, these tests were conducted using a HPE Apollo 2000 server running CentOS Linux 7 and which had a Intel Haswell CPU and 128 GB of RAM.

For experiments described in §6.1–§6.5 we utilized a test set consisting of the instances examined in Belotti et al. [BSW12] and Boland et al. [BCS14; BCS15b]. The former contained 30 instances with 60 variables and 60 constraints (`Belotti60`) and 30 instances with 80 variables and 80 constraints (`Belotti80`). The latter had 5 instances for each of the three types `Boland80`, `Boland160`, and `Boland320` (we do not solve instances with less than 60 constraints or variables due to their relative ease), and 4 instances for each of the three types `Boland16`, `Boland25`, and `Boland50`.[2]



(a) Instance from the `Belotti60` set.

(b) Instance from the `Boland16` set.

Figure 6 **Pareto set and boundary of $\mathcal{L}_0$ for the two instance families.**

Figure 6 depicts the Pareto set and boundary of $\mathcal{L}_0$ for one instance from each of the two instance classes. Note the following structural differences displayed in the two figures

1. The relative gap between the Pareto set and boundary of $\mathcal{L}_0$ is greater in Figure 6a than in Figure 6b.

2. The relative gap between connected subsets of the Pareto set is greater in Figure 6a than in Figure 6b.

3. The overall number of solutions present in the Pareto set is greater in Figure 6b than in Figure 6a.

We found that the above differences were typical for these instance families. This provides some insight into the differences in performance seen for these two instances families through the rest of this section. Note, in particular, the difference in duality gaps seen

[2] These are labelled this way to maintain consistency with the way other instances are labeled although the respective total number of variables and constraints is approximately 800, 1250 and 2500.

in Experiments 1, 3, and 5 as well as the difference in number of nodes processed when utilizing $\mathcal{OS}$ gap splitting in Experiment 4.

Our final set of experiments are described in §6.6, where we opted to generate a more difficult test set. For this purpose, we created biobjective variants of instances from MIPLIB 2017 [Gle+19] that were feasible, mixed-integer, marked easy, and contained at most 1000 decision variables. For each such instance, we generated two secondary objective functions and discarded instances for which: (i) the Pareto set was a singleton, or (ii) the second objective was unbounded, or (iii) the MILP associated with either f_1 or f_2 took over 8 hours to solve.

The computational tests with our BB had a maximum solution time of 8 hours. For each instance, we recorded the computation time in seconds, the number of nodes explored in our BB tree, and the % duality gap computed after the root node was processed. We report average values of these numbers for the `Belotti*` instances, which we recall are thirty of each type.

We began our tests by turning off all nonessential features of our BB procedure, and then sequentially turning on various features to test their impact on the overall procedure. If a particular feature of our BB procedure was deemed effective in reducing the overall effort required to solve instances of BOMILP, this feature was left on for the remainder of the tests, otherwise it was turned back off.

Our original implementation included a variety of features which did not prove useful in either reducing the overall BB time or the number of explored nodes. For the sake of space, in the sections that follow we focus only on features that proved useful. We briefly note some of these ideas here to motivate future research into them. Most of our fruitless features involved adding various cutting planes to the problem formulation. Note that we are not referring to CPLEX default cut generation – this was left on and did prove useful. Instead, we are referring to: (i) attempts to add user-generated cuts from discovered disjunctions, and (ii) attempts to use CPLEX default cut generation *at each node* and add the discovered cuts as local cuts. Other attempted features included checks for early termination of Fathoming Rule 3 and the generation of $\mathcal{ND}(\mathcal{L}_s)$. Each of these provided inconsistent results, reducing BB time for some problems but increasing it for others. Hence, both were abandoned in the end.

6.1. Presolve Techniques

Table 1 contains the results of our first computational experiment. Note that in for this test we utilized PreprocesingMethod2 with ρ set to zero.

Table 1 Experiment 1 – Measuring the impact of presolve techniques

Instance	All Off			Duality Fixing			Singleton Columns			Dominating Columns		
	Time	Nodes	$\bar{\mathcal{G}}_0$	Time	Nodes	$\bar{\mathcal{G}}_0$	Time	Nodes	$\bar{\mathcal{G}}_0$	Time	Nodes	$\bar{\mathcal{G}}_0$
Belotti60 (30)	4	77	53	4	77	53	4	77	53	4	77	53
Belotti80 (30)	11	96	52	11	96	52	11	96	52	11	96	52
Boland80	16	507	46	15	520	46	18	507	46	16	507	46
	9	267	23	6	268	37	10	267	23	9	267	23
	26	668	17	21	689	17	26	668	17	27	668	17
	16	531	19	11	415	19	17	531	19	17	531	19
	14	465	22	11	400	18	14	465	22	13	465	22
	16	**488**	**25**	**13**	**458**	**27**	**17**	**488**	**25**	**17**	**488**	**25**
Boland160	430	3133	13	387	2944	13	444	3133	13	445	3133	13
	564	2543	12	483	2437	12	549	2543	12	544	2543	12
	241	1781	13	276	2303	20	233	1781	13	239	1781	13
	782	3646	15	814	3768	15	763	3646	15	777	3646	15
	302	2021	17	291	2086	13	291	2021	17	301	2021	17
	464	**2625**	**14**	**450**	**2708**	**15**	**456**	**2625**	**14**	**461**	**2625**	**14**
Boland320	13019	10862	10	16403	17004	63	13009	10862	10	13355	10862	10
	22572	15924	8	22102	17575	8	22931	15924	8	22306	15924	8
	22006	14403	9	24181	21072	75	21820	14403	9	22153	14403	9
	21831	16990	10	22486	18319	12	21837	16990	10	20380	16990	10
	15981	13597	9	13840	12569	9	15277	13597	9	14204	13597	9
	19082	**14355**	**9**	**19802**	**17308**	**33**	**18975**	**14355**	**9**	**18480**	**14355**	**9**
Boland16	2	32	5	1	32	5	2	32	5	2	32	5
	3	49	11	2	47	11	2	49	11	2	49	11
	7	125	27	5	123	27	7	125	27	6	125	27
	10	183	25	8	183	25	10	183	25	10	183	25
	5	**97**	**17**	**4**	**96**	**17**	**5**	**97**	**17**	**5**	**97**	**17**
Boland25	14	162	14	13	183	14	13	162	14	14	162	14
	25	283	15	22	289	15	26	283	15	25	283	15
	40	429	13	33	422	13	39	429	13	40	429	13
	43	437	20	41	466	20	44	437	20	43	437	20
	31	**328**	**16**	**27**	**340**	**16**	**31**	**328**	**16**	**31**	**328**	**16**
Boland50	395	1343	14	341	1409	14	397	1343	14	397	1343	14
	754	1952	17	606	1890	17	766	1952	17	772	1952	17
	1427	2593	9	1249	2437	9	1382	2593	9	1357	2593	9
	1740	3386	15	615	1622	15	1754	3386	15	1702	3386	15
	1079	**2319**	**14**	**703**	**1840**	**14**	**1074**	**2319**	**14**	**1057**	**2319**	**14**

Notice from Table 1 that the results for duality fixing show the opposite pattern for the `Boland320` instances than for all other instances. This is due to the fact that, for an unknown reason, fixing several variables during presolve had a negative impact on preprocessing, causing many fewer solutions to be discovered during this phase and therefore having an overall negative impact on the rest of the BB procedure. We felt though that the positive impact duality fixing had on the other instances sets warranted leaving this feature on for the remainder of our tests. Also observe from Table 1 that the exploitation of neither singleton nor dominating columns had any significant impact on the overall BB procedure. We found that this was mainly due to the fact that there were very few occurrences of either of these types of columns. We opted to turn off the exploitation of singleton columns for the remainder of our tests, but we left on the exploitation of dominating columns. Our reasoning here was that singleton columns have no impact on BB that extends beyond presolve, while dominating columns result in disjunctions from which we can generate global cutting planes. Hence, we left on the exploitation of dominating columns in order to test the impact of generating these cuts in later tests.

6.2. Preprocessing

In our next test we examined the impact of the two preprocessing techniques discussed in Section 3.1, as well as a hybrid method we derived as a combination of the two presented procedures. In our initial implementation of this test we used each of these methods with ρ assigned each integer value in $[0, 5]$. Recognize from Algorithms 1 and 2 that each of the proposed preprocessing procedures are designed so that the total number of Pareto solutions computed should have a positive correlation with the value of ρ. We determined that PROPROCESINGMETHOD1 performed poorly for $\rho \leq 1$, PROPROCESINGMETHOD2 performed poorly for $\rho \geq 2$ and the hybrid method performed poorly in general. Hence, we do not report results for these procedures. We also discovered that the impact of ρ on overall solution time varied with the size of the instance solved. As a result, we also implemented modified preprocessing procedures in which the value of ρ is automatically computed as a function of the size of an instance. For each family of instance, the average CPU required to complete BB after employing each of the aforementioned preprocessing strategies is reported in Table 2. We note that in Table 2 $\rho = v$ indicates that ρ was automatically computed as a function of instance size.

Table 2 Experiment 2 – Measuring the impact of preprocessing techniques

Instance	Time (s)							
	PreprocessingMethod1					PreprocessingMethod2		
	$\rho=2$	3	4	5	v	$\rho=0$	1	v
Belotti60 (30)	4	4	8	8	4	4	5	5
Belotti80 (30)	11	11	18	18	11	11	12	12
Boland80	10	11	10	10	11	15	16	15
	5	7	6	7	8	6	7	7
	20	18	17	18	22	21	26	25
	16	17	16	17	16	11	12	13
	7	6	7	7	11	11	12	12
	12	**12**	**11**	**12**	**13**	**13**	**14**	**14**
Boland160	388	299	218	219	298	401	383	393
	300	265	266	263	335	487	516	506
	177	144	125	125	158	282	291	287
	549	552	541	557	548	816	880	862
	171	185	158	158	443	302	280	290
	317	**289**	**262**	**264**	**356**	**458**	**470**	**467**
Boland320	11036	8619	6398	6232	9561	16480	16544	16636
	15099	16278	16210	16142	14963	22319	21181	21246
	9433	10421	9615	9840	10675	24151	21878	21788
	14379	16642	16446	16427	16253	22837	24763	24384
	10303	10706	10779	10811	10602	14422	14440	14449
	12050	**12533**	**11890**	**11891**	**12411**	**20042**	**19761**	**19701**
Boland16	1	2	2	3	1	2	3	3
	2	2	3	3	2	2	3	3
	5	5	6	6	5	5	7	7
	8	9	10	9	8	8	10	9
	4	**4**	**5**	**5**	**4**	**4**	**6**	**5**
Boland25	11	10	10	10	12	14	16	17
	18	19	18	18	18	23	26	25
	22	22	22	23	24	31	37	37
	50	49	50	53	52	40	44	43
	25	**25**	**25**	**26**	**27**	**27**	**31**	**31**
Boland50	278	293	196	198	335	342	354	356
	633	546	456	464	663	583	689	678
	990	1110	743	708	945	1250	1848	1852
	599	2217	1325	1382	1325	625	2054	2001
	625	**1042**	**680**	**688**	**817**	**700**	**1236**	**1222**

Observe from Table 2 that although variants of PreprocessingMethod2 performed well for smaller instances, the same is not true for larger instances. PreprocessingMethod1, on the other hand, performed quite well on all instances. Notice, however, that values of ρ near two performed quite well for small instances while values near five performed extremely poorly. On the other hand, for larger instances values of ρ near five

seem to outperform almost every other procedure. Due to the consistent performance of the variant of PREPROCESSINGMETHOD1 with $\rho = 2$, we opted to use this approach for the remainder of our tests.

6.3. Probing and Pareto Branching

The next test we performed was designed to examine the utility of the variable probing procedure used directly after preprocessing and at each node prior to branching, and the Pareto branching that we perform when $\mathcal{OS}$ fathoming results in disjoint feasible regions of $\mathcal{OS}$. The results of this experiment are given in Table 3.

Observe from Table 3 that when utilizing probing directly after preprocessing, in many cases the total CPU time and number of nodes processed increased. Surprisingly, however, performing the same probing procedure prior to branching at each node had an extremely positive impact on the overall performance of BB, significantly lowering total CPU time and the number of explored nodes. We also found that Pareto branching had an overall positive impact on BB performance. For the remainder of our tests we opted to cease probing directly after preprocessing, but to still employ probing during branching as well as Pareto branching.

6.4. Exploiting $\mathcal{OS}$ Gaps and Comparing with Triangle Splitting

We now present the results of an experiment designed to test the performance of our BB procedure against that of the triangle splitting (TS) method of [BCS15b]. For this experiment we solved all the same instances we used in our previous tests and employed two variants of our BB procedure, one in which we utilized the $\mathcal{OS}$ splitting procedure we discussed in Section 5.2 and one in which we utilized our standard implementation. The results of this test are given in Table 4. Our standard BB procedure outperformed the triangle splitting method on all but one set of instances, while our $\mathcal{OS}$ splitting procedure outperformed the triangle splitting method on all sets of instances except one. Also recognize that the total CPU times associated with our $\mathcal{OS}$ splitting procedure are always comparable with those of our standard procedure. We point out that there were many more substantial gaps between solutions to exploit after preprocessing for the Belotti* instances than for the Boland* instances. This is the reason that there is a drastic reduction in total number of nodes processed when using $\mathcal{OS}$ splitting on the `Belotti*` instances but not

Table 3 Experiment 3 – Measuring the impact of Probing and Pareto branching

Instance	All Off			Initial Probing			Probing in Branching		Pareto Branching	
	Time	Nodes	$\bar{\mathcal{G}}_0$	Time	Nodes	$\bar{\mathcal{G}}_0$	Time	Nodes	Time	Nodes
`Belotti60` (30)	4	72	48	4	74	49	3	48	4	72
`Belotti80` (30)	11	98	49	11	97	49	8	62	10	94
`Boland80`	10	368	12	10	366	12	5	160	11	388
	5	256	35	6	254	35	5	210	7	260
	20	647	19	20	647	19	7	283	17	621
	14	598	45	16	581	45	9	393	15	526
	7	302	18	7	284	18	5	157	7	332
	11	**434**	**26**	**12**	**426**	**26**	**6**	**241**	**11**	**425**
`Boland160`	393	3185	19	349	2815	19	125	1082	259	2394
	309	1713	20	338	1743	20	122	625	290	1948
	171	1466	5	178	1433	5	91	651	149	1551
	547	2982	8	547	3016	8	201	1249	488	3595
	167	1196	28	168	1154	28	78	570	153	1447
	318	**2108**	**16**	**316**	**2032**	**16**	**123**	**835**	**268**	**2187**
`Boland320`	10951	10391	6	11061	10673	6	3099	3882	7120	8292
	14601	12827	6	15038	12954	6	5012	5329	11358	12004
	9402	7626	12	9316	7598	12	3173	3380	7571	8072
	14065	12161	6	14542	12528	6	5583	5679	11685	13181
	9991	9900	5	9850	9930	5	2664	3462	6555	8380
	11802	**10581**	**7**	**11962**	**10737**	**7**	**3906**	**4346**	**8858**	**9986**
`Boland16`	1	29	5	1	28	5	1	28	1	47
	2	54	12	2	56	12	1	43	2	63
	5	128	42	5	124	42	3	104	6	163
	7	165	12	7	168	12	5	129	9	199
	4	**94**	**18**	**4**	**94**	**18**	**3**	**76**	**4**	**118**
`Boland25`	11	157	32	11	159	32	7	130	10	175
	18	343	36	18	337	36	12	259	23	445
	23	370	64	29	505	64	15	284	26	379
	50	764	76	52	765	76	33	545	38	580
	25	**409**	**52**	**28**	**442**	**52**	**17**	**305**	**24**	**395**
`Boland50`	278	1501	33	304	1660	33	165	1063	292	1831
	614	2318	44	749	2799	44	499	1862	585	2857
	948	2966	22	1101	3367	22	600	2188	704	2949
	559	2083	60	2038	5349	60	1001	2583	438	2135
	600	**2217**	**40**	**1048**	**3294**	**40**	**566**	**1924**	**505**	**2443**

the `Boland*` instances. We also did a parallel implementation of the $\mathcal{OS}$ splitting procedure and observed some reduction in the CPU times, which suggests that parallelising this procedure can further improve the BB algorithm.

Table 4 Experiment 4 – Measuring the impact of $\mathcal{OS}$ Gap Splitting

Instance	Standard BB Time	Standard BB Nodes	BB with $\mathcal{OS}$ Gaps Time	BB with $\mathcal{OS}$ Gaps Nodes	TS Time
Belotti60 (30)	3	49	4	33	9
Belotti80 (30)	7	64	8	44	20
Boland80	6	205	5	205	44
	3	203	3	138	29
	7	326	7	261	46
	7	300	6	262	48
	3	165	3	165	32
	5	**240**	**5**	**206**	**40**
Boland160	79	914	85	886	320
	97	734	105	749	335
	67	692	60	668	267
	180	1631	188	1626	677
	56	691	48	573	258
	96	**932**	**97**	**900**	**371**
Boland320	2048	3391	2055	3371	3800
	3213	4568	3333	4743	6219
	1981	3135	1957	3164	5035
	3239	5429	3328	5385	5421
	1755	3461	1912	3697	4293
	2447	**3997**	**2517**	**4072**	**4954**
Boland16	1	39	1	39	4
	2	47	1	47	5
	2	94	4	128	10
	5	133	5	133	13
	3	**78**	**3**	**87**	**8**
Boland25	6	137	6	119	19
	14	325	9	215	30
	13	258	16	347	39
	22	397	24	433	51
	14	**279**	**14**	**279**	**35**
Boland50	158	1156	137	961	159
	374	2058	306	1754	262
	484	2240	371	1795	346
	990	3843	977	3369	475
	502	**2324**	**448**	**1970**	**311**

6.5. Approximations of the Pareto Set

Boland et al. [BCS15b] measured the time it takes the Triangle Splitting method to compute an approximate Pareto set having the property that the hypervolume gap between valid primal and dual bounds implied by this approximate set is less than 2%. We repeat this experiment for our BB procedure, though we note that the primal and dual bounds we

utilize are significantly different than those used in [BCS15b]. We measure this gap directly after the completion of our preprocessing procedure, and then each time 25 nodes are processed during BB. We cease the procedure if: (i) BB terminates with the true Pareto set, or (ii) the hypervolume gap is less than 2%. In this experiment we also report Hausdorff gap measurements, as described in Section 5.3. Additionally, for comparison we include certain results as reported in [BCS15b].

The results of this experiment are displayed in Table 5 from which we make several observations. For the majority of the Boland* instances, the hypervolume gap is already less than 2% after preprocessing, before BB even begins. This is evidence that these instances are relatively easy. Recall Figure 6, and notice that for the `Boland80` instance the boundary of the dual bound at the root node is very close to the Pareto set. This is further evidence of the ease of these instances. In contrast to this, notice from Table 5 that for the Belotti* instances, it takes over 75% of the total BB time in order to obtain a hypervolume gap of less than 2%. We note that Table 5 also shows that the triangle splitting method is able to determine an approximate solution with a hypervolume gap of less than 2% in less time, relative to the total solution time.

6.6. MIPLIB Instances

Due to the successful results we obtained using our BB procedure on instances from the literature, we designed our final set of tests to measure the performance of our procedure on a more realistic set of instances. For this we utilized a set of single objective MILP instances available from the MIPLIB 2017 library [Gle+19]. We chose only instances that were feasible, mixed-integer, contained at most 1000 total decision variables, and were marked easy. For each instance, we generated two secondary objective functions as follows:

(r) For each $i \in \{1, \dots, m+n\}$ the coefficient c_i^2 is randomly generated using the uniform distribution over the closed interval $[-|\max_i c_i^1|, |\max_i c_i^1|]$.

(n) We set $c_i^2 = -c_i^1$.

After generation of these instances we did some preliminary testing and discarded instances for which: (i) the Pareto set was a singleton, or (ii) the second objective was unbounded, or (iii) the MILP associated with either f_1 or f_2 took over 8 hours to solve. In the end, 104 instances remained for final testing (2 each, originating from 52 single objective MILP instances).

Table 5 Experiment 5 – Obtaining approximate Pareto sets

	Preprocessing		Standard BB Until $\mathcal{HV}_{s^*} \leq 2\%$						TS
Instance	$\mathcal{HV}_0$	$\bar{\mathcal{G}}_0$	Time	% Time	Nodes	% Nodes	$\mathcal{HV}_{s^*}$	$\bar{\mathcal{G}}_{s^*}$	% Time
Belotti60 (30)	21.7	62.4	3	100	49	98	0.2	4.9	–
Belotti80 (30)	25.7	66.8	7	100	62	98	0.2	3.3	–
Boland80	1.6	12.3	1	18	0	0	1.6	12.3	12
	3.5	34.7	3	74	100	49	2.0	16.3	9
	2.9	19.0	2	31	25	8	1.8	10.9	4
	49.0	67.1	5	74	225	75	1.2	10.7	6
	2.2	18.2	1	47	25	15	1.2	11.2	7
	11.8	**30.3**	**2**	**49**	**75**	**29**	**1.6**	**12.3**	**7.6**
Boland160	2.6	18.6	16	21	75	8	1.2	8.8	2.30
	1.9	20.0	8	9	0	0	1.9	20.0	3.85
	1.2	4.7	4	6	0	0	1.2	4.7	1.50
	0.8	7.9	10	5	0	0	0.8	7.9	0.61
	9.2	28.4	20	35	150	22	1.8	8.4	2.90
	3.1	**15.9**	**12**	**15**	**45**	**6**	**1.4**	**10.0**	**2.23**
Boland320	1.1	6.4	52	3	0	0	1.1	6.4	0.21
	0.5	5.9	82	3	0	0	0.5	5.9	0.23
	0.5	12.3	78	4	0	0	0.5	12.3	0.26
	0.5	5.9	80	2	0	0	0.5	5.9	0.23
	0.4	5.5	72	4	0	0	0.4	5.5	0.22
	0.6	**7.2**	**73**	**3**	**0**	**0**	**0.6**	**7.2**	**0.23**
Boland16	0.6	5.3	1	68	0	0	0.6	5.3	–
	1.2	11.9	0	26	0	0	1.2	11.9	–
	3.1	42.3	2	74	25	27	1.0	18.6	–
	2.2	12.3	2	34	25	19	1.6	11.4	–
	1.7	**18.0**	**1**	**50**	**13**	**11**	**1.1**	**11.8**	–
Boland25	4.0	32.2	5	82	75	55	1.2	9.7	–
	67.3	79.4	13	94	175	54	1.8	19.9	–
	83.0	87.8	7	55	75	29	1.5	28.5	–
	91.2	93.7	11	50	100	25	1.9	22.0	–
	61.4	**73.3**	**9**	**70**	**106**	**41**	**1.6**	**20.0**	–
Boland50	2.6	33.3	15	10	25	2	1.9	28.9	–
	3.8	44.3	152	41	325	16	2.0	28.5	–
	1.9	21.8	6	1	0	0	1.9	21.8	–
	26.6	75.1	191	19	275	7	1.7	24.0	–
	8.7	**43.6**	**91**	**18**	**156**	**6**	**1.9**	**25.8**	–

A primary reason for generating this additional set of instances is the relative ease with which single objective MILPs were solved throughout the solution process, both during the execution of triangle splitting and our BB, when using the previously considered test sets. As such, in our first analysis of these new instances we set a variety of node limits on single objective MILPs solved during our BB (other than the two specified on line 4 of Algorithm 7, of course). By limiting the number of nodes processed during each single

objective MILP solve, we hoped to increase the speed of the overall BB procedure while still being able to exploit useful dual bound information at each node. For initial tests, we set node limits of 10, 10^2, 10^3, 10^4, and ∞ and compared the overall solution time for BB on all 104 instances, with a maximum execution time of 8 hours. Surprisingly, the best performing node limits were 10^4 and ∞. Hence, we opted to leave the single objective MILP node limit off for the remainder of our analysis. We did note, however, that on some instances, single objective MILPs took significant time to solve even when relatively few nodes were explored in order to do so. Thus, we opted to solve each instance again, this time with an overall time limit imposed when solving each single objective MILP. For this test we utilized time limits of 15, 30, 45, 60, 300, 1800, and ∞ seconds. In this case, the limits that appeared to produce the best results were 30 and 300 seconds, with 300 seconds having a slight advantage. We therefore imposed a single objective MILP time limit of 300 seconds when conducting our final round of tests.

Table 6 gives the results of this experiment, where the two lines for each instance correspond to the (r) and (n) methods, respectively, for generating the second objective function. Of the 104 instances considered, 49 were solved in under 8 hours by the original BB implementation, 52 by the $\mathcal{OS}$ splitting BB variant, and 64 by the triangle splitting method. Additionally, there were 11 instances which were solved in under 8 hours by at least one version of BB, but not by the triangle splitting method, and 21 instances solved in under 8 hours by the triangle splitting method, but not by a BB procedure. In all, the results display comparable performance between the BB approaches and the triangle splitting method, though for instances in which there was a relative difference in performance, it was generally large. From what we can tell, these discrepancies in performance seem to stem from overall structure of the Pareto set in $\mathcal{OS}$. In particular, triangle splitting appears have superior performance on instances for which either: (i) the total number of Pareto solutions is small, or (ii) most Pareto solutions are supported, particularly if all Pareto solutions lie along a single line segment in $\mathcal{OS}$. Both of these properties can be observed in the Pareto sets of instances beginning with "mik," for example. On the other hand, BB appears to have superior performance on instances for which either: (i) the total number of Pareto solutions is large, or (ii) a relatively large percentage of Pareto solutions are unsupported. We also note that occasionally numerical issues caused early termination of BB when solving instances for which all Pareto solutions fall on a single line segment in $\mathcal{OS}$. In

Table 6 Experiment 6 – Solution time (sec.) for biobjective instances generated from MIPLIB 2017.

	Branch-and-bound		Triangle Splitting		Branch-and-bound		Triangle Splitting
Instance	Standard	Gap Splitting		Instance	Standard	Gap Splitting	
22433	5.53	3.93	5.36	neos-1425699	*	0.04	*
	21.31	59.43	139.40		*	*	*
23588	60.69	50.60	91.47	neos-1430701	5797.07	27292.20	2086.53
	24616.54	25092.16	790.16		1.87	1.90	2.64
assign1-5-8	25599.20	25104.12	1867.39	neos-1442119	*	*	22552.6
	2035.07	2031.20	1415.81		326.48	328.68	1534.18
b-ball	0.60	0.31	0.06	neos17	*	*	25302.70
	0.01	0.01	0.02		*	*	*
beavma	4178.27	3615.45	*	neos-3610041-iscar	731.45	142.51	8.14
	*	*	*		3.37	3.42	3.91
blend2	5232.48	5274.62	*	neos-3610051-istra	2002.65	880.83	37.81
	*	*	28405.80		*	0.91	6.34
ci-s4	*	*	*	neos-3610173-itata	12251.52	2.72	75.74
	13511.01	15171.00	*		*	*	25.90
dcmulti	725.25	728.54	173.88	neos-3611447-jijia	2934.49	1354.50	37.68
	*	*	0.76		*	*	19.07
exp-1-500-5-5	*	*	*	neos-3611689-kaihu	398.39	9.27	53.70
	5.59	5.56	21.36		*	*	18.41
fastxgemm-n2r6s0t2	*	*	178.48	neos5	285.47	286.68	265.46
	1419.83	1414.09	0.6		2613.94	3046.41	273.85
flugpl	0.48	0.44	1.46	Neos-5192052-neckar	0.79	0.79	*
	0.02	0.02	0.04		9.72	9.71	0.06
gen	72.61	74.01	15.57	Nexp-50-20-1-1	*	145.16	1318.04
	*	*	0.32		1.01	*	28.08
gr4x6	0.69	0.56	1.12	noswot	407.97	198.68	261.77
	*	*	5.29		272.75	195.12	171.7
ic97_potential	*	*	*	nsa	*	1813.48	*
	21411.78	21597.36	*		*	*	*
ic97_tension	*	*	*	opt1217	0.21	0.21	0.15
	9.10	9.12	53.19		0.10	0.09	0.02
k16x240b	*	*	*	prod1	*	19240.07	978.34
	390.86	664.73	*		20.29	23.34	27.52
markshare_4_0	3665.90	112.33	9841.90	prod2	*	*	25214.90
	27.75	28.49	768.88		283.95	283.36	183.98
markshare_5_0	*	*	27098.70	qiu	863.70	1002.68	18267.50
	*	*	6767.01		*	*	*
mas74	27118.70	20170.72	19182.20	r50x360	*	*	*
	486.88	489.42	*		921.03	*	*
mas76	*	*	*	ran12x21	*	*	*
	*	*	*		*	*	*
mik_250_20_75_1	*	*	*	ran13x13	8732.94	8420.79	*
	*	*	20.57		*	*	*
mik_250_20_75_2	*	*	*	ran14x18-disj-8	*	*	*
	*	*	14.62		*	*	*
mik_250_20_75_3	*	*	*	rout	1191.12	1095.00	2795.34
	*	*	15.60		*	*	*
mik_250_20_75_4	*	*	*	sp150x300d	*	*	*
	*	*	139.28		*	*	39.27
mik_250_20_75_5	*	*	*	timtab1	*	*	*
	*	*	17.69		*	*	*
misc07	*	*	3816.02	timtab1CUTS	*	*	*
	*	*	614.38		*	*	*

particular, for some such instances a cutting plane was generated along this line segment in $\mathcal{OS}$ after which BB execution ceased. This phenomenon was observed on the "markshare_4_0" instance, for example, which explains the apparent difference in performance patterns between BB and triangle splitting for the "markshare" instances.

7. Concluding Remarks

In this paper, we have introduced a new BB method for solving BOMILP with general integers. For each component of single objective BB, we presented procedure(s) for extending this component to the biobjective setting. We have also conducted numerous computational experiments. The first several experiments provide insight into the usefulness of each of the algorithms we proposed. The final few experiments compare the performance of our BB procedure and the triangle splitting method [BCS15b]. Our BB procedure outperforms the triangle splitting method on instances from literature, and performs comparably on large, challenging instances that were developed in this paper.

Most of the algorithms proposed by us have, in theory, straightforward generalizations to the multiobjective case (MOMILPs). However, having an implementable correct BB for MOMILPs is far from a trivial extension of this work. We point out some important questions that need to be answered in this regard.

7.1. Extension to multiobjective MILP

Correct node fathoming is what makes a BB algorithm a correct and exact method. Fathoming by bound dominance is how fathoming mostly occurs in BB. For BOMILP, the bound sets are two-dimensional polyhedra. This greatly simplifies checking bound dominance for BOMILPs since given two line segments, or piecewise linear curves in general, in $\mathbb{R}^2$, one can easily identify the dominated portion through pairwise comparisons. The data structure [ABG18] stores nondomimated line segments and efficiently checks if a new line segment is dominated by what is currently stored. This enabled the node processing step in this paper to perform fathoming efficiently. Bound sets for MOMILP are higher-dimensional polyhedra and hence one will require an even more sophisticated data structure to store these sets. Since the local dual bound set at each node is a polyhedron and the global primal bound is a finite union of polyhedra, checking dominance requires checking containment of polyhedra, whose complexity depends on their respective representations, and also computing the set difference between the primal and dual bound sets.

The set resulting from this set difference would be nonconvex, in general, which begs the question: is there a straightforward way to represent this nonconvex set as a union of polyhedra whose relative interiors are disjoint? All in all, fathoming and storing nondominated regions for a MOMILP is even more nontrivial. Once these obstacles are overcome, the BB proposed in this paper should extend to a implementable BB for MOMILPs.

References

[AW13] T. Achterberg and R. Wunderling. "Mixed integer programming: Analyzing 12 years of progress". In: *Facets of Combinatorial Optimization: Festschrift for Martin Grötschel*. Ed. by M. Jünger and G. Reinelt. Springer, 2013, pp. 449–481.

[ABG18] N. Adelgren, P. Belotti, and A. Gupte. "Effecient storage of Pareto points in biobjective mixed integer programming". In: *INFORMS Journal on Computing* 30.2 (2018), pp. 324–338.

[BJV13] C. Bazgan, F. Jamain, and D. Vanderpooten. "On the number of nondominated points of a multicriteria optimization problem". In: *Discrete Applied Mathematics* 161.18 (2013), pp. 2841–2850.

[BJV15] C. Bazgan, F. Jamain, and D. Vanderpooten. "Approximate Pareto sets of minimal size for multi-objective optimization problems". In: *Operations Research Letters* 43.1 (2015), pp. 1–6.

[BSW12] P. Belotti, B. Soylu, and M. M. Wiecek. *A Branch-and-Bound Algorithm for Biobjective Mixed-Integer Programs*. Tech. rep. Clemson University, Dec. 2012.

[BSW16] P. Belotti, B. Soylu, and M. M. Wiecek. "Fathoming rules for biobjective mixed integer linear programs: Review and extensions". In: *Discrete Optimization* 22.Part B (2016), pp. 341–363.

[BGP09] J.-F. Bérubé, M. Gendreau, and J.-Y. Potvin. "An exact ε-constraint method for bi-objective combinatorial optimization problems: Application to the Traveling Salesman Problem with Profits". In: *European Journal of Operational Research* 194.1 (2009), pp. 39–50.

[BP12] V. Blanco and J. Puerto. "A new complexity result on multiobjective linear integer programming using short rational generating functions". In: *Optimization Letters* 6.3 (2012), pp. 537–543.

[BCS14] N. Boland, H. Charkhgard, and M. Savelsbergh. "The triangle splitting method for biobjective mixed integer programming". In: *Integer Programming and Combinatorial Optimization: IPCO 2014*. Ed. by J. Lee and J. Vygen. Vol. 8494. Lecture Notes in Computer Science. Springer, Cham, 2014, pp. 162–173.

[BCS15a] N. Boland, H. Charkhgard, and M. Savelsbergh. "A criterion space search algorithm for biobjective integer programming: The balanced box method". In: *INFORMS Journal on Computing* 27.4 (2015), pp. 735–754.

[BCS15b] N. Boland, H. Charkhgard, and M. Savelsbergh. "A criterion space search algorithm for biobjective mixed integer programming: The triangle splitting method". In: *INFORMS Journal on Computing* 27.4 (2015), pp. 597–618.

[BCS16a] N. Boland, H. Charkhgard, and M. Savelsbergh. "The L-shape search method for triobjective integer programming". In: *Mathematical Programming Computation* 8.2 (2016), pp. 217–251.

[BCS16b] N. Boland, H. Charkhgard, and M. Savelsbergh. "The Quadrant Shrinking Method: A simple and efficient algorithm for solving tri-objective integer programs". In: *European Journal of Operational Research* (2016).

[BKR17] R. S. Burachik, C. Y. Kaya, and M. Rizvi. "A new scalarization technique and new algorithms to generate Pareto fronts". In: *SIAM Journal on Optimization* 27.2 (2017), pp. 1010–1034.

[DHK09] J. A. De Loera, R. Hemmecke, and M. Köppe. "Pareto optima of multicriteria integer linear programs". In: *INFORMS Journal on Computing* 21.1 (2009), pp. 39–48.

[Ehr05] M. Ehrgott. *Multicriteria optimization*. Springer, 2005.

[Ehr06] M. Ehrgott. "A discussion of scalarization techniques for multiple objective integer programming". In: *Annals of Operations Research* 147.1 (2006), pp. 343–360.

[EG07] M. Ehrgott and X. Gandibleux. "Bound sets for biobjective combinatorial optimization problems". In: *Computers & Operations Research* 34.9 (2007), pp. 2674–2694.

[ER08] M. Ehrgott and S. Ruzika. "Improved ε-constraint method for multiobjective programming". In: *Journal of Optimization Theory and Applications* 138.3 (2008), pp. 375–396.

[GNE19] S. L. Gadegaard, L. R. Nielsen, and M. Ehrgott. "Bi-objective Branch-and-Cut Algorithms Based on LP Relaxation and Bound Sets". In: *INFORMS Journal on Computing* 31.4 (2019), pp. 790–804.

[Gam+15] G. Gamrath, T. Koch, A. Martin, M. Miltenberger, and D. Weninger. "Progress in presolving for mixed integer programming". In: *Mathematical Programming Computation* 7.4 (2015), pp. 367–398.

[Gle+19] A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. M. Christophel, K. Jarck, T. Koch, J. Linderoth, M. Lübecke, H. D. Mittelmann, D. Ozyurt, T. K. Ralphs, D. Salvagnin, and Y. Shinano. *MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library*. Technical Report. Optimization Online, July 2019.

[Gra+14] F. Grandoni, R. Ravi, M. Singh, and R. Zenklusen. "New approaches to multi-objective optimization". In: *Mathematical Programming* 146.1-2 (2014), pp. 525–554.

[JLS12] N. Jozefowiez, G. Laporte, and F. Semet. "A generic branch-and-cut algorithm for multiobjective optimization problems: Application to the multilabel traveling salesman problem". In: *INFORMS Journal on Computing* 24.4 (2012), pp. 554–564.

[KY83] G. Kiziltan and E. Yucaoğlu. "An algorithm for multiobjective zero-one linear programming". In: *Management Science* 29.12 (1983), pp. 1444–1453.

[KH82] D. Klein and E. Hannan. "An algorithm for the multiple objective integer linear programming problem". In: *European Journal of Operational Research* 9.4 (1982), pp. 378–385.

[LLS14] M. Leitner, I. Ljubić, and M. Sinnl. "A computational study of exact approaches for the bi-objective prize-collecting steiner tree problem". In: *INFORMS Journal on Computing* 27.1 (2014), pp. 118–134.

[Lei+16] M. Leitner, I. Ljubić, M. Sinnl, and A. Werner. "ILP heuristics and a new exact method for bi-objective 0/1 ILPs: Application to FTTx-network design". In: *Computers & Operations Research* 72 (2016), pp. 128–146.

[LK13] B. Lokman and M. Köksalan. "Finding all nondominated points of multi-objective integer programs". In: *Journal of Global Optimization* 57 (2013), pp. 347–365.

[Mar01] A. Martin. "General mixed integer programming: Computational issues for branch-and-cut algorithms". In: *Computational Combinatorial Optimization: Optimal or Provably Near-Optimal Solutions*. Ed. by M. Jünger and D. Naddef. Vol. 2241. Lecture Notes in Computer Science. Springer, 2001, pp. 1–25.

[MD05] G. Mavrotas and D. Diakoulaki. "Multi-criteria branch and bound: A vector maximization algorithm for Mixed 0-1 Multiple Objective Linear Programming". In: *Applied Mathematics and Computation* 171.1 (2005), pp. 53–71.

[MF13] G. Mavrotas and K. Florios. "An improved version of the augmented ε-constraint method (AUGMECON2) for finding the exact pareto set in multi-objective integer programming problems". In: *Applied Mathematics and Computation* 219.18 (2013), pp. 9652–9669.

[Mor+16] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell. "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning". In: *Discrete Optimization* 19 (2016), pp. 79–102.

[ÖK10] Ö. Özpeynirci and M. Köksalan. "An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs". In: *Management Science* 56.12 (2010), pp. 2302–2315.

[PT19] S. N. Parragh and F. Tricoire. "Branch-and-bound for bi-objective integer programming". In: *INFORMS Journal on Computing* 31.4 (2019), pp. 805–822.

[PG17] A. Przybylski and X. Gandibleux. "Multi-objective branch and bound". In: *European Journal of Operational Research* 260.3 (2017), pp. 856–872.

[PGE10] A. Przybylski, X. Gandibleux, and M. Ehrgott. "A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives". In: *Discrete Optimization* 7.3 (2010), pp. 149–165.

[RSW06] T. K. Ralphs, M. J. Saltzman, and M. M. Wiecek. "An improved algorithm for solving biobjective integer programs". In: *Annals of Operations Research* 147.1 (2006), pp. 43–70.

[RW07] J. O. Royset and R. K. Wood. "Solving the bi-objective maximum-flow network-interdiction problem". In: *INFORMS Journal on Computing* 19.2 (2007), pp. 175–184.

[RW05] S. Ruzika and M. M. Wiecek. "Approximation methods in multiobjective programming". In: *Journal of Optimization Theory and Applications* 126.3 (2005), pp. 473–501.

[Say03] S. Sayin. "A procedure to find discrete representations of the efficient set with specified coverage errors". In: *Operations Research* 51.3 (2003), pp. 427–436.

[Say00] S. Sayın. "Measuring the quality of discrete representations of efficient sets in multiple objective mathematical programming". In: *Mathematical Programming* 87.3 (2000), pp. 543–560.

[SS08] F. Sourd and O. Spanjaard. "A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem". In: *INFORMS Journal on Computing* 20.3 (2008), pp. 472–484.

[Soy15] B. Soylu. "Heuristic approaches for biobjective mixed 0–1 integer linear programming problems". In: *European Journal of Operational Research* 245.3 (2015), pp. 690–703.

[SVS13] M. Stanojević, M. Vujošević, and B. Stanojević. "On the cardinality of the nondominated set of multi-objective combinatorial optimization problems". In: *Operations Research Letters* 41.2 (2013), pp. 197–200.

[SAD14] T. Stidsen, K. A. Andersen, and B. Dammann. "A branch and bound algorithm for a class of biobjective mixed integer programs". In: *Management Science* 60.4 (2014), pp. 1009–1032.

[Vin+13] T. Vincent, F. Seipp, S. Ruzika, A. Przybylski, and X. Gandibleux. "Multiple objective branch and bound for mixed 0-1 linear programming: Corrections and improvements for the biobjective case". In: *Computers & Operations Research* 40.1 (2013), pp. 498–509.

[Vis+98] M. Visée, J. Teghem, M. Pirlot, and E. Ulungu. "Two-phases Method and Branch and Bound Procedures to Solve the Biobjective Knapsack Problem". In: *Journal of Global Optimization* 12 (2 1998), pp. 139–155.

[Zit+03] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. Da Fonseca. "Performance assessment of multiobjective optimizers: An analysis and review". In: *IEEE Transactions on Evolutionary Computation* 7.2 (2003), pp. 117–132.