Multi-objective Optimization Based Algorithms for Solving Mixed Integer Linear Minimum Multiplicative Programs

Vahid Mahmoodian
Department of Industrial and Management Systems Engineering, University of South Florida, Tampa, FL 33620, USA
Hadi Charkhgard
Department of Industrial and Management Systems Engineering, University of South Florida, Tampa, FL 33620, USA
Yu Zhang
Department of Civil and Environmental Engineering, University of South Florida, Tampa, FL 33620, USA

We present two new algorithms for a class of single-objective non-linear optimization problems, the so-called Mixed Integer Linear minimum Multiplicative Programs (MIL-mMPs). This class of optimization problems has a desirable characteristic: a MIL-mMP can be viewed as a special case of the problem of optimization over the efficient set in multi-objective optimization. The proposed algorithms exploit this characteristic and solve any MIL-mMP from the viewpoint of multi-objective optimization. A computational study on 960 instances demonstrates that the proposed algorithms outperform a generic-purpose solver, SCIP, by a factor of more than 10 on many instances. We numerically show that selecting the best algorithm among our proposed algorithms highly depends on the class of instances used. Specifically, since one of the proposed algorithms is a decision space search algorithm and the other one is a criterion space search algorithm, one can significantly outperform the other depending on the dimension of decision space and criterion space. Although it is possible to linearize some instances of MIL-mMPs, we show that a commercial solver, CPLEX, struggles to directly solve such linearized instances because linearization introduces additional constraints and binary decision variables.

Key words: multi-objective optimization; minimum multiplicative programming; optimization over the efficient set; decision space search algorithm; criterion space search algorithm
History: Submitted on November 14, 2019

1. Introduction

Multi-objective optimization provides decision-makers with a complete view of the trade-offs between their objective functions that are attainable by feasible solutions. It is thus a critical tool in engineering, where competing goals must often be considered and balanced when making decisions. Combined with the fact that many problems can be formulated as mixed integer linear programs, the development of fast and reliable multi-objective mixed

integer programming solvers has significant benefits for problem solving in industry and government. Consequently, it is not surprising that in the last decade, many researchers have focused on developing effective algorithms for solving multi-objective mixed integer linear programs, such as the ones conducted by Ehrgott and Gandibleux (2007), Özlen and Azizoğlu (2009), Özpeynirci and Köksalan (2010), Dächert et al. (2012), Lokman and Köksalan (2013), Eusébio et al. (2014), Kirlik and Sayın (2015), Soylu and Yıldız (2016), Boland et al. (2017b), and Przybylski and Gandibleux (2017).

While promising, a non-trivial (but interesting) question that has received less attention in the literature is whether multi-objective optimization methods can be useful for solving single-objective optimization problems? In recent years, a few studies have explored this question and have shown the superiority of multi-objective optimization based techniques (compared to the existing methods) on solving two classes of single-objective optimization problems including minimum Multiplicative Programs (mMPs; see for instance Shao and Ehrgott (2014, 2016)) and Maximum Multiplicative Programs (MMPs; see for instance Charkhgard et al. (2018) and Saghand et al. (2019)). A mMP is the problem of the form,

$$\min \left\{ \prod_{i=1}^{p} y_i(\boldsymbol{x}) : \ \boldsymbol{x} \in \mathcal{X}, \ \boldsymbol{y}(\boldsymbol{x}) \geq \boldsymbol{0} \right\}, \tag{1}$$

where $\mathcal{X} \subseteq \mathbb{R}^n$ represents the set of feasible solutions and it is assumed to be bounded. Also, $\boldsymbol{y}(\boldsymbol{x}) := \big(y_1(\boldsymbol{x}), \ldots, y_p(\boldsymbol{x})\big)$ is a vector of linear functions. As an aside, throughout this article, vectors are always column-vectors and are denoted in bold fonts. It is assumed that the optimal objective value of a mMP is strictly positive. We note that if $\mathcal{X}$ is defined by a set of linear constraints, the problem is referred to as Linear mMP (L-mMP). If in a L-mMP, all decision variables are integers, the problem is referred to as Integer Linear mMP (IL-mMP). Finally, if in a L-mMP, some but not all decision variables are integers, the problem is referred to as Mixed Integer Linear mMP (MIL-mMP).

It is worth mentioning that if in a mMP, one changes 'min' to 'max', a MMP will be constructed. Although MMPs and mMPs look almost the same, they are very different in terms of their computational complexity. Specifically, L-MMPs (meaning linear MMPs) are polynomially solvable but L-mMPs are NP-hard (Charkhgard et al. 2018, Shao and Ehrgott 2016). This negative result may have contributed to the fact that (to the best of our knowledge), all (multi-objective based) methodological studies on mMPs have focused

only on solving mMPs involving no integer decision variables. For example, Konno and Kuno (1992) showed how L-mMPs when $p = 2$ can be solved by combining the parametric simplex method and any standard convex minimization procedure. Benson and Boger (1997) proposed a heuristic method for solving L-mMPs. In another study, Kuno (2001) proposed a branch-and-bound algorithm for solving L-mMPs. Kim et al. (2007) presented an outcome-space outer approximation algorithm for L-mMPs. Recently, Shao and Ehrgott (2016) proposed two objective-space algorithms for solving L-mMPs. It is worth mentioning that there are also some other studies such as those conducted by Van Thoai (1991), Kuno et al. (1993), Benson (1999), Gao et al. (2010), Oliveira and Ferreira (2008), Gao et al. (2010), and Shao and Ehrgott (2014) on developing algorithms for mMPs involving no integer decision variables (but not necessarily L-mMPs) that interested readers may refer to.

In light of the above, there is a clear gap in the literature about developing effective multi-objective optimization based techniques for solving mMPs involving integer decision variables. Consequently, the goal of this study is to develop new exact multi-objective optimization based algorithms that can solve any MIL-mMP by just solving a number of single-objective (mixed integer) linear programs. In other words, we attempt to develop new algorithms that can exploit the power of commercial single-objective mixed integer linear programming solvers such as IBM ILOG CPLEX for solving a MIL-mMP (which is a non-linear optimization problem).

We note that developing more effective techniques for MIL-mMPs can have significant impacts on problem solving in practice as MIL-mMPs (and in general mMPs) have several applications in different fields of study including (but not limited to) bond portfolio management, economic analysis, and VLSI chip design (Benson and Boger 2000) as well as conservation planning or natural resource management (Nicholson and Possingham 2006). For example, in conservation planning, a typical goal is to preserve biodiversity. For doing so, solution $\boldsymbol{x} \in \mathcal{X}$ typically represents a subset of parcels (or sites) that should be selected for protection within a geographical region. Also, $y_i(\boldsymbol{x})$ typically represents the probability of species $i \in \{1, \ldots, p\}$ being extinct if solution $\boldsymbol{x}$ is implemented. In other words, the objective function of a mMP in conservation planning represents the probability of all species being extinct and the goal is to minimize it (to preserve biodiversity).

The proposed exact algorithms in this study are developed based on a critical observation that an optimal solution of Problem (1) is an *efficient* or *Pareto-optimal* solution, i.e., a solution in which it is impossible to improve the value of one objective without making the value of at least one other objective worse, of the following multi-objective optimization problem,

$$\min \Big\{ y_1(\boldsymbol{x}), \dots, y_p(\boldsymbol{x}) : \ \boldsymbol{x} \in \mathcal{X}, \ \boldsymbol{y}(\boldsymbol{x}) \geq \boldsymbol{0} \Big\}. \tag{2}$$

Specifically, by minimizing the function $\prod_{i=1}^{p} y_i(\boldsymbol{x})$ over the set of efficient solutions of Problem (2), an optimal solution of Problem (1) can be obtained. We prove this observation in Section 2 but similar proofs can be found in Benson and Boger (1997), Charkhgard et al. (2018), Saghand et al. (2019) and Shao and Ehrgott (2014, 2016). Overall, this critical observation indicates that Problem (1) can be viewed and solved as a special case of the problem of *optimization over the efficient set* in multi-objective optimization. It is worth noting that, in optimization over the efficient set, the goal is to compute an optimal solution directly, i.e., without enumerating all efficient solutions if possible (Jorge 2009, Sierra Altamiranda and Charkhgard 2019, Boland et al. 2017a, Sayın 2000, Benson 1984).

The main contribution of our research is that we employ the above observation and develop two new algorithms for solving MIL-mMPs. The first proposed algorithm is a decision space search algorithm, i.e., an algorithm that works in the space of decision variables of a multi-objective optimization problem. The algorithm performs similar to the classical branch-and-bound algorithm for solving single-objective mixed integer linear programs. Specifically, the algorithm solves a number of L-mMPs in order to solve a MIL-mMP. To solve each L-mMP, existing multi-objective optimization algorithms can be employed and they solve only a number of linear programs to compute an optimal solution. In other words, our first algorithm solves only (single-objective) linear programs for solving each MIL-mMP and we use the power of commercial linear programming solvers to solve each one. For improving the performance of our first algorithm, we also propose some enhancement techniques and show their effectiveness in a computational study.

Our second proposed algorithm is a criterion space search algorithm, i.e., an algorithm that works in the space of objective values of a multi-objective optimization problem. The proposed algorithm solves a number of (single-objective) mixed integer linear programs (rather than linear programs) to solve each MIL-mMP. So, the main advantage of the second algorithm is that it uses the power of commercial mixed integer linear programming

solvers. To show the effectiveness of the proposed algorithms, we conduct a comprehensive computational study on 960 instances and compare the performance of our proposed algorithms with a generic-purpose solver, SCIP. The results show that our proposed algorithms can outperform SCIP by a factor of more than 10 on many instances in terms of solution time. By a detailed comparison, we show that for many instances with $p = 2$, our proposed criterion space search algorithm outperforms the proposed decision space search algorithm by a factor of more than 8 in terms of solution time. However, as p increases, our criterion space algorithm starts to struggle. Hence, we show that for instances with $p = 4$, the proposed decision space search algorithm significantly outperforms our proposed criterion space search algorithm in terms of optimality gap obtained within the imposed time limit. We also numerically show that as the number of integer decision variables increases, the proposed decision space search algorithm starts to struggle. Hence, for instances with $p = 3$, if the number of integer decision variables is large, the proposed criterion space search is the best choice. Otherwise, the proposed decision space search algorithm is the best approach. Furthermore, we show that linearizing the objective function of IL-mMPs, i.e., when no continuous variable exists, is possible but it will result in large single-objective integer linear programs. So, in the computational study, we show that a commercial solver, CPLEX, struggles to solve such linearized instances. Hence, using the proposed algorithms is a better choice.

The rest of the paper is organized as follows. In Section 2, some preliminaries about MIL-mMPs are introduced. In Section 3, the details of the proposed decision space search algorithm are provided. In Section 4, the proposed criterion space search algorithm is explained. In Section 5, a comprehensive computational study is presented. Finally, in Section 6, some concluding remarks are given.

2. Preliminaries

A Mixed Integer Linear minimum Multiplicative Program (MIL-mMP) can be stated as follows,

$$\begin{aligned} \min \ & \prod_{i=1}^{p} y_i \\ \text{s.t. } & \boldsymbol{y} = C\boldsymbol{x} + \boldsymbol{d} \\ & A\boldsymbol{x} \geq \boldsymbol{b} \\ & \boldsymbol{x}, \boldsymbol{y} \geq \boldsymbol{0}, \quad \boldsymbol{x} \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_i}, \quad \boldsymbol{y} \in \mathbb{R}^p, \end{aligned} \tag{3}$$

where n_c and n_i denote the number of continuous and integer decision variables, respectively. Also, C is a $p \times n$ matrix where $n := n_c + n_i$ and p shows the number of variables in the objective function. Finally, $\boldsymbol{d}$ is a vector of length p, A is an $m \times n$ matrix, and $\boldsymbol{b}$ is a vector of length m.

We refer to the set $\mathcal{X} := \{\boldsymbol{x} \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_i} : A\boldsymbol{x} \geq \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0}\}$ as *the feasible set in the decision space* and to the set $\mathcal{Y} := \{\boldsymbol{y} \in \mathbb{R}^p : \boldsymbol{x} \in \mathcal{X},\ \boldsymbol{y} = C\boldsymbol{x} + \boldsymbol{d},\ \boldsymbol{y} \geq \boldsymbol{0}\}$ as *the feasible set in the criterion space*. We assume that $\mathcal{X}$ is bounded (which implies that $\mathcal{Y}$ is compact) and the optimal objective value of the problem is strictly positive, i.e., $\boldsymbol{0} \notin \mathcal{Y}$. Throughout this article, we refer to $\boldsymbol{x} \in \mathcal{X}$ as a *feasible solution* and to $\boldsymbol{y} \in \mathcal{Y}$ as a *feasible point* ($\boldsymbol{y}$ is the image of $\boldsymbol{x}$ in the criterion space).

Definition 1. A feasible solution $\boldsymbol{x} \in \mathcal{X}$ is called *efficient*, if there is no other $\boldsymbol{x}' \in \mathcal{X}$ such that $\boldsymbol{y}' \leq \boldsymbol{y}$ and $\boldsymbol{y}' \neq \boldsymbol{y}$ where $\boldsymbol{y} := C\boldsymbol{x} + \boldsymbol{d}$ and $\boldsymbol{y}' := C\boldsymbol{x}' + \boldsymbol{d}$. If $\boldsymbol{x}$ is efficient, then $\boldsymbol{y}$ is called a *nondominated point*. The set of all efficient solutions is denoted by $\mathcal{X}_E$. The set of all nondominated points is denoted by $\mathcal{Y}_N$ and referred to as the *nondominated frontier*.

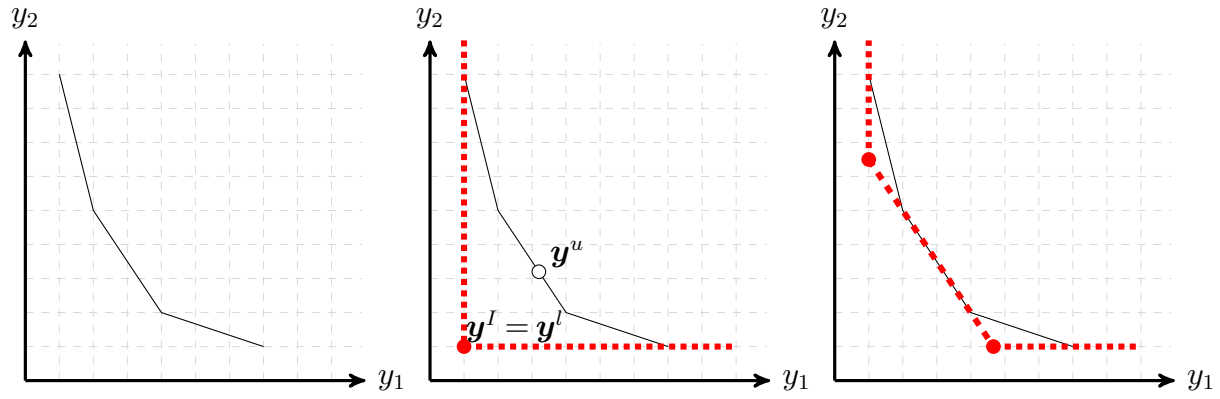
Proposition 1. *An optimal solution of Problem* (3)*, denoted by* $\boldsymbol{x}^*$*, is an efficient solution and therefore its corresponding image in the criterion space, denoted by* $\boldsymbol{y}^*$ *where* $\boldsymbol{y}^* := C\boldsymbol{x}^* + \boldsymbol{d}$*, is a nondominated point.*

Proof. Let $\boldsymbol{x}^*$ be an optimal solution of Problem (3) but not an efficient solution. By definition, this implies that there must exist a feasible solution $\boldsymbol{x} \in \mathcal{X}$ such that it dominates $\boldsymbol{x}^*$. In other words, we must have that $\boldsymbol{y} \leq \boldsymbol{y}^*$ and $\boldsymbol{y} \neq \boldsymbol{y}^*$ where $\boldsymbol{y} := C\boldsymbol{x} + \boldsymbol{d}$ and $\boldsymbol{y}^* = C\boldsymbol{x}^* + \boldsymbol{d}$. Also, by assumptions of Problem (3), we know that $\boldsymbol{y}, \boldsymbol{y}^* > \boldsymbol{0}$. Therefore, the inequalities $0 < \prod_{i=1}^{p} y_i < \prod_{i=1}^{p} y_i^*$ must hold. However, this immediately implies that $\boldsymbol{x}^*$ is not an optimal solution (a contradiction). □

Proposition 1 implies that Problem (3) is equivalent to $\min_{\boldsymbol{y} \in \mathcal{Y}_N} \prod_{i=1}^{p} y_i$ and this is precisely optimization over the efficient set. This observation indicates that developing multi-objective optimization techniques for solving MIL-mMPs is possible and this idea will be explored in this paper.

3. A decision space search algorithm

In this section, we propose a new decision space search algorithm for solving MIL-mMPs. As mentioned in the Introduction, in the field of multi-objective optimization, decision space search algorithms are referred to solution approaches that work in the space of decision



(a) The nondominated frontier of a L-mMP

(b) Critical points in the first iteration

(c) The relaxed set at the end of the first iteration

Figure 1 **An illustration of the workings of the method proposed by Shao and Ehrgott (2016) on a L-mMP with** $p = 2$

variables. The proposed approach is similar to the classical branch-and-bound algorithm for solving single-objective mixed integer linear programs. The underlying idea of our approach is to effectively employ the techniques that can solve any L-mMP within a branch-and-bound framework to solve MIL-mMPs. To the best of our knowledge, developing decision space search algorithms has not yet explored for MIL-mMPs. However, it is recently studied (see Saghand et al. (2019)) in the literature of MIL-MMPs but only for instances with $p = 2$. Overall, our proposed branch-and-bound framework is similar to the one proposed by Saghand et al. (2019). The main difference is that a completely different method should be employed for computing dual bounds which can be obtained by solving L-mMPs. In the rest of this section, we first review one of the fastest algorithms for solving L-mMPs. After that, the proposed branch-and-bound framework is explained. Finally, a few enhancement techniques are introduced.

3.1. A method for solving L-mMPs

Although our proposed algorithm can employ any method for solving L-mMPs, in this section we briefly review the one that we found to perform the best within our branch-and-bound framework (during the course of our research). The method is developed by Shao and Ehrgott (2016) and is one of the fastest and (relatively) easy-to-implement algorithms for solving L-mMPs. This method acts as the default solution approach for solving L-mMPs at the heart of our proposed branch-and-bound framework.

The method of Shao and Ehrgott (2016) maintains a global lower bound and a global upper bound and it terminates whenever these two values converge. The underlying idea

of the method is to use Proposition 1 for solving a L-mMP. Hence, the method attempts to find an optimal point of the L-mMP by searching over its nondominated frontier. It is worth mentioning that the nondominated frontier of a feasible L-mMP is known to be a convex curve and consists of only some plane/line segments. An illustration of the nondominated frontier of a L-mMP when $p = 2$ can be found in Figure 1a. Overall, the method of Shao and Ehrgott (2016) is iterative but in order to start, it requires to be initialized by the so-called *ideal* point of the L-mMP, i.e., the imaginary point in the criterion space that has the minimum possible value for each objective. Note that to compute the ideal point, denoted by $\boldsymbol{y}^I$, p linear programs should be solved since $y_i^I := \min_{\boldsymbol{y} \in \mathcal{Y}^R} y_i$ for $i = 1, \ldots, p$ where $\mathcal{Y}^R := \{\boldsymbol{y} \in \mathbb{R}^p : \boldsymbol{x} \in \mathcal{X}^R,\ \boldsymbol{y} = C\boldsymbol{x} + \boldsymbol{d},\ \boldsymbol{y} \geq \boldsymbol{0}\}$ and $\mathcal{X}^R := \{\boldsymbol{x} \in \mathbb{R}^n : A\boldsymbol{x} \geq \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0}\}$. An illustration of the ideal point, denoted by $\boldsymbol{y}^I$, when $p = 2$ can be found in Figure 1b. Observe that the set of nondominated points of the L-mMP is a subset of $S := \{\boldsymbol{y} \in \mathbb{R}^p : \boldsymbol{y} \geq \boldsymbol{y}^I\}$. We call S as the relaxed set and it plays a key role in the method of Shao and Ehrgott (2016).

In each iteration, the method computes a global dual/lower bound for the optimal objective value of the L-mMP using the relaxed set. Specifically, it can be shown that an extreme point of the relaxed set, denoted by $\boldsymbol{y}^l$, that has the minimum value for the multiplicative function, i.e., the objective function of the L-mMP, is a global dual bound. For example, in the first iteration, the only extreme point in the relaxed set is the ideal point, i.e., $\boldsymbol{y}^l = \boldsymbol{y}^I$, and $\prod_{i=1}^p y_i^I$ is obviously a global lower/dual bound (see Figure 1b). As an aside, to compute the vertices of the relaxed set, the algorithm proposed by Chen et al. (1991) can be employed. In each iteration, the method also uses $\boldsymbol{y}^l$ to compute a new feasible point, denoted by $\boldsymbol{y}^u$ with $\boldsymbol{y}^l \leq \boldsymbol{y}^u$. In order to do so, the following linear program needs to be solved that attempts to find a feasible point with the minimum Chebyshev distance from $\boldsymbol{y}^l$,

$$\boldsymbol{y}^u \in \arg\min_{\boldsymbol{y} \in \mathcal{Y}^R} \big\{ z \in \mathbb{R} : z \geq 0 \text{ and } y_i - y_i^l \leq z\ \forall i \in \{1, \ldots, p\} \text{ and } y_i^l \leq y_i\ \forall i \in \{1, \ldots, p\} \big\}, \quad (4)$$

where z is a continuous decision variable that captures the Chebyshev distance. An illustration of $\boldsymbol{y}^u$ can be found in Figure 1b. Note that whenever a new nondominated point is found, it can provide an upper bound for the optimal objective value of the L-mMP because it is a feasible point. So, the global upper/primal bound needs to be updated after computing a new nondominated point in each iteration. Moreover, it can be shown

(see Shao and Ehrgott (2016)) that the inequality $\boldsymbol{\lambda}^\intercal \boldsymbol{y} \geq \boldsymbol{\lambda}^\intercal \boldsymbol{y}^u$ will not remove any non-dominated point of the L-mMP where $\boldsymbol{\lambda}$ is a vector of length p. For each $i \in \{1, \ldots, p\}$, λ_i captures the dual value associated with constraint $y_i - y_i^l \leq z$ in Problem (4). So, the inequality $\boldsymbol{\lambda}^\intercal \boldsymbol{y} \geq \boldsymbol{\lambda}^\intercal \boldsymbol{y}^u$ will be added at the end of each iteration for the purpose of restricting/updating the relaxed set. An illustration of the relaxed set after adding the cut can be found in Figure 1c.

3.2. The proposed branch-and-bound framework

Since the method proposed by Shao and Ehrgott (2016) is frequently used in our proposed algorithm, we first introduce a notation/operation for it. Specifically, in our proposed algorithm, whenever we want to use the method proposed by Shao and Ehrgott (2016), we provide a lower bound vector, denoted by $\boldsymbol{l} \in \mathbb{R}^n$, and an upper bound vector, denoted by $\boldsymbol{u} \in \mathbb{R}^n$, for $\boldsymbol{x}$. So, we introduce the operation LB-FINDER$(\boldsymbol{l}, \boldsymbol{u})$ when we want to solve the following L-mMP using the method proposed by Shao and Ehrgott (2016),

$$\begin{aligned} \min \ & \prod_{i=1}^{p} y_i \\ \text{s.t. } & \boldsymbol{y} = D\boldsymbol{x} + \boldsymbol{d} \\ & A\boldsymbol{x} \geq \boldsymbol{b} \\ & \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \\ & \boldsymbol{x}, \boldsymbol{y} \geq \boldsymbol{0}, \quad \boldsymbol{x} \in \mathbb{R}^n, \quad \boldsymbol{y} \in \mathbb{R}^p, \end{aligned} \tag{5}$$

This operation simply returns an optimal solution and an optimal point of Problem (5) denoted by $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$. Note that if $\tilde{\boldsymbol{x}}$ =null (or equivalently $\tilde{\boldsymbol{y}}$=null) then Problem (5) is infeasible. Next, we provide the details of our proposed algorithm which is designed to report an optimal solution $\boldsymbol{x}^*$ (or its image in the criterion $\boldsymbol{y}^*$) of a MIL-mMP when it terminates.

The algorithm maintains a queue of nodes, denoted by TREE. The algorithm also maintains a global upper bound, denoted by $\mathrm{G_{UB}}$, and a global lower bound, denoted by $\mathrm{G_{LB}}$. At the beginning, the algorithm sets $\mathrm{G_{UB}} = +\infty$ and $\mathrm{G_{LB}} = -\infty$. Moreover, the algorithm initializes $(\boldsymbol{l}, \boldsymbol{u})$ with $(\boldsymbol{0}, +\boldsymbol{\infty})$. To initialize the queue, the algorithm first calls LB-FINDER$(\boldsymbol{l}, \boldsymbol{u})$ to compute $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$. If the integrality conditions hold, i.e., $\tilde{\boldsymbol{x}} \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_i}$, then an optimal solution is found. So, in that case, the algorithm terminates after setting $(\boldsymbol{x}^*, \boldsymbol{y}^*) = (\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$. Otherwise, by definition, LB-FINDER$(\boldsymbol{l}, \boldsymbol{u})$ has computed a dual

bound, and so the algorithm sets $\mathrm{G_{LB}} = \prod_{i=1}^{p} \tilde{y}_i$ and initializes the queue by $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}, \boldsymbol{l}, \boldsymbol{u})$. The algorithm then explores the queue as long as it is nonempty and $\mathrm{G_{UB}} - \mathrm{G_{LB}} \geq \varepsilon_1$ and $\frac{\mathrm{G_{UB}} - \mathrm{G_{LB}}}{\mathrm{G_{UB}}} \geq \varepsilon_2$, where $\varepsilon_1, \varepsilon_2 \in (0, 1)$ are the user-defined absolute and relative optimality gap tolerances, respectively. Next, we explain how each element of the queue is explored.

Algorithm 1: A Decision Space Search Algorithm

Input: A feasible instance of Problem (3)
Queue.create(TREE)
$(\boldsymbol{l}, \boldsymbol{u}) \leftarrow (\boldsymbol{0}, +\boldsymbol{\infty})$
$\mathrm{G_{LB}} \leftarrow -\infty$; $\mathrm{G_{UB}} \leftarrow +\infty$
$(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) \leftarrow$ LB-FINDER$(\boldsymbol{l}, \boldsymbol{u})$
if $\tilde{\boldsymbol{x}} \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_i}$ **then**
    $(\boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow (\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$
    $\mathrm{G_{UB}} \leftarrow \prod_{i=1}^{p} \tilde{y}_i$
else
    TREE.*add*$\big((\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}, \boldsymbol{l}, \boldsymbol{u})\big)$
    $\mathrm{G_{LB}} \leftarrow \prod_{i=1}^{p} \tilde{y}_i$
while *not Queue.empty*(TREE) *&* $\mathrm{G_{UB}} - \mathrm{G_{LB}} \geq \varepsilon_1$ *&* $\frac{\mathrm{G_{UB}} - \mathrm{G_{LB}}}{\mathrm{G_{UB}}} \geq \varepsilon_2$ **do**
    TREE.*PopOut*$\big((\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{u})\big)$
    $j \leftarrow$ BRANCHING-INDEX$(\boldsymbol{x})$
    $(\boldsymbol{l}^1, \boldsymbol{u}^1) \leftarrow (\boldsymbol{l}, \boldsymbol{u})$; $l_j^1 \leftarrow \lceil x_j \rceil$
    $(\boldsymbol{l}^2, \boldsymbol{u}^2) \leftarrow (\boldsymbol{l}, \boldsymbol{u})$; $u_j^2 \leftarrow \lfloor x_j \rfloor$
    $(\boldsymbol{x}^1, \boldsymbol{y}^1) \leftarrow$ LB-FINDER$(\boldsymbol{l}^1, \boldsymbol{u}^1)$
    if $(\boldsymbol{x}^1, \boldsymbol{y}^1) \neq$*(null,null)* **then**
        if $\boldsymbol{x}^1 \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_i}$ *&* $\prod_{i=1}^{p} y_i^1 < \mathrm{G_{UB}}$ **then**
            $(\boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow (\boldsymbol{x}^1, \boldsymbol{y}^1)$
            $\mathrm{G_{UB}} \leftarrow \prod_{i=1}^{p} y_i^1$
        else if $\mathrm{G_{UB}} - \prod_{i=1}^{p} y_i^1 \geq \varepsilon_1$ *&* $\frac{\mathrm{G_{UB}} - \prod_{i=1}^{p} y_i^1}{\mathrm{G_{UB}}} \geq \varepsilon_2$ **then**
            TREE.*add*$\big((\boldsymbol{x}^1, \boldsymbol{y}^1, \boldsymbol{l}^1, \boldsymbol{u}^1)\big)$
    $(\boldsymbol{x}^2, \boldsymbol{y}^2) \leftarrow$ LB-FINDER$(\boldsymbol{l}^2, \boldsymbol{u}^2)$
    if $(\boldsymbol{x}^2, \boldsymbol{y}^2) \neq$*(null,null)* **then**
        if $\boldsymbol{x}^2 \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_i}$ *&* $\prod_{i=1}^{p} y_i^2 < \mathrm{G_{UB}}$ **then**
            $(\boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow (\boldsymbol{x}^2, \boldsymbol{y}^2)$
            $\mathrm{G_{UB}} \leftarrow \prod_{i=1}^{p} y_i^2$
        else if $\mathrm{G_{UB}} - \prod_{i=1}^{p} y_i^2 \geq \varepsilon_1$ *&* $\frac{\mathrm{G_{UB}} - \prod_{i=1}^{p} y_i^2}{\mathrm{G_{UB}}} \geq \varepsilon_2$ **then**
            TREE.*add*$\big((\boldsymbol{x}^2, \boldsymbol{y}^2, \boldsymbol{l}^2, \boldsymbol{u}^2)\big)$
    Update $\mathrm{G_{LB}}$
return $\boldsymbol{x}^*, \boldsymbol{y}^*$

In each iteration, the algorithm pops out an element of the queue and denotes it by $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{u})$. Note that when an element is popped out from the queue then that element does not exist in the queue anymore. Of course elements (or nodes) can be popped out in different orders and it is known that employing the right *node selection* strategy can impact the solution time of branch-and-bound algorithms in practice. During the course of this research, we implemented five well-known node selection strategies (see Saghand et al. (2019) for details) including *depth-first search*, *best-bound search*, *two-phase method*, *best-expected bound*, and *best estimate*. The most promising node selection strategy for our proposed framework is the best estimate and hence it is used as the default setting of our algorithm in the rest of this paper.

The algorithm next selects an index of a decision variable that was supposed to take an integer value but it currently has fractional value in solution $\boldsymbol{x}$. This operation is denoted by Branching-Index($\boldsymbol{x}$) and its output is denoted by j. Similar to the node selection strategies, there are several *variable selection* strategies in practice that can be used for the operation Branching-Index($\boldsymbol{x}$). During the course of this research, we implemented four well-known variable selection strategies (see Saghand et al. (2019) for details) including *random branching*, *most infeasible branching*, *pseudo-cost branching*, and *reliability branching*. The most promising variable selection strategy for our proposed framework is pseudo-cost branching and hence it is used as the default setting of our algorithm in the rest of this paper.

Next, the algorithm generates two new lower bound and upper bound vectors, denoted by $(\boldsymbol{l}^1, \boldsymbol{u}^1)$ and $(\boldsymbol{l}^2, \boldsymbol{u}^2)$. The algorithm sets $l_i^1 = l_i$ for all $i \in \{1, \dots, n\} \backslash \{j\}$, $l_j^1 = \lceil x_j \rceil$, and $\boldsymbol{u}^1 = \boldsymbol{u}$. Similarly, the algorithm sets $u_i^2 = u_i$ for all $i \in \{1, \dots, n\} \backslash \{j\}$, $u_j^2 = \lfloor x_j \rfloor$, and $\boldsymbol{l}^2 = \boldsymbol{l}$. The algorithm first explores $(\boldsymbol{l}^1, \boldsymbol{u}^1)$. This implies that the algorithm calls LB-Finder($\boldsymbol{l}^1, \boldsymbol{u}^1$) to compute $(\boldsymbol{x}^1, \boldsymbol{y}^1)$. Again we note that if LB-Finder fails to find a feasible solution then we have that $\boldsymbol{x}^1$ =null and $\boldsymbol{y}^1$ =null. Therefore, in that case, the algorithm will skip the following steps and proceed to explore $(\boldsymbol{l}^2, \boldsymbol{u}^2)$. Otherwise, the algorithm first checks whether the integrality conditions hold, i.e., $\boldsymbol{x}^1 \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_i}$, and $\prod_{i=1}^{p} y_i^1 < \mathrm{G_{UB}}$. If that is the case then a new and better global lower bound is found and so the algorithm will set $(\boldsymbol{x}^*, \boldsymbol{y}^*) = (\boldsymbol{x}^1, \boldsymbol{y}^1)$ and $\mathrm{G_{UB}} = \prod_{i=1}^{p} y_i^1$. Otherwise, the algorithm checks whether $\mathrm{G_{UB}} - \prod_{i=1}^{p} y_i^1 \geq \varepsilon_1$ and $\frac{\mathrm{G_{UB}} - \prod_{i=1}^{p} y_i^1}{\mathrm{G_{UB}}} \geq \varepsilon_2$. If that is the case then the algorithm will add

$(\boldsymbol{x}^1, \boldsymbol{y}^1, \boldsymbol{l}^1, \boldsymbol{u}^1)$ to be explored further in the future because it is possible to find better feasible solutions by exploring that element.

After exploring $(\boldsymbol{l}^1, \boldsymbol{u}^1)$, $(\boldsymbol{l}^2, \boldsymbol{u}^2)$ should be explored similarly. Therefore, the algorithm will explore it and then before starting the next iteration it will update $\mathrm{G_{LB}}$. The minimum value of $\prod_{i=1}^{p} y_i$ among all nodes in the queue defines the new global lower bound. A detailed description of the proposed decision space search algorithm can be found in Algorithm 1.

3.3. Enhancements

In this section, we introduce two enhancement techniques that can be used to possibly improve the performance of Algorithm 1.

Enhancement-I (Preprocessing) According to Proposition 1, optimal points of a MIL-mMP must be nondominated points. Therefore, any nondominated point can potentially result in computing a good primal/upper bound for a MIL-mMP. So, one trivial enhancement technique is to compute a few nondominated points before starting Algorithm 1 and then feeding the best primal bound obtained from them as the initial $\mathrm{G_{UB}}$ to Algorithm 1. In order to generate nondominated points, we use the so-called weighted sum operation. This operation attempts to compute a nondominated point $\bar{\boldsymbol{y}}$ by solving the following optimization problem,

$$\bar{\boldsymbol{y}} \in \arg\min \{\sum_{i=1}^{p} \lambda_i y_i : y \in \mathcal{Y}\},$$

where $\lambda_1, \ldots, \lambda_p > 0$ are user-defined weights. We denote the weighted sum operation by $\mathrm{WSO}(\boldsymbol{\lambda})$. As an aside, it is known that the weighted sum operation always returns a nondominated point but not all nondominated points can be necessarily found using the weighted sum operation (even by considering all possible values of $\lambda_1, \ldots, \lambda_p$) for non-convex optimization problems (Ehrgott 2005, Aneja and Nair 1979).

In the literature of multi-objective optimization, there exists several exact algorithms for effectively choosing values for the vector $\boldsymbol{\lambda}$ (see for instance Özpeynirci and Köksalan (2010)). However, such approaches can be very time consuming for MIL-mMPs and we do not want to spend the valuable computational time on generating the nondominated points. So, instead, we propose a (preprocessing) heuristic approach which is motivated from the study of Pal and Charkhgard (2019) for computing some nondominated points. Specifically, we impose a time limit for generating nondominated points in our heuristic approach. It is worth mentioning that during the course of this research, we found that

setting the time limit to $0.01n$ seconds, where n is the number of decision variables, results in the best performance for our algorithm (in our test instances). So, the default value for the time limit of our heuristic approach is $0.01n$ in the remaining of this paper.

The proposed heuristic is an iterative approach that terminates whenever the time limit reaches. In each iteration, the components of the weight vector $\boldsymbol{\lambda}$ will be generated randomly from the standard normal distribution, denoted by $\text{RandNormal}(0,1)$. However, we use the absolute value of the generated random numbers. Also, for convenience, we normalize the components of each weight vector to assure that the summation of its components will be equal to one. After generating $\boldsymbol{\lambda}$ in each iteration, the proposed heuristic calls $\text{WSO}(\boldsymbol{\lambda})$ to compute a nondominated point $\bar{\boldsymbol{y}}$. Afterwards, the proposed heuristic updates the global upper bound value, i.e., G_{UB}, if $\prod_{i=1}^{p} \bar{y}_i$ is better than its current value.

Algorithm 2: A preprocessing approach

Input: A feasible instance of Problem (3)
$\text{G}_{\text{UB}} \leftarrow +\infty$
while *Time* $\leq$ *TimeLimit* **do**
  foreach $i \in \{1,\dots,p\}$ **do**
    $\lambda_i \leftarrow |\,\text{RandNormal}(0,1)|$
  foreach $i \in \{1,\dots,p\}$ **do**
    $\lambda_i \leftarrow \frac{\lambda_i}{\sum_{j=1}^{p} \lambda_j}$
  $\bar{\boldsymbol{y}} \leftarrow \text{WSO}(\boldsymbol{\lambda})$
  if $\prod_{i=1}^{p} \bar{y}_i < \text{G}_{\text{UB}}$ **then**
    $\text{G}_{\text{UB}} \leftarrow \prod_{i=1}^{p} \bar{y}_i$
  $\mathcal{Y} \leftarrow \mathcal{Y} \cap \{\boldsymbol{y} \in \mathbb{R}^p : \sum_{i=1}^{p} \lambda_i y_i \geq \lambda_i \bar{y}_i\}$
return $\mathcal{Y}, \text{G}_{\text{UB}}$

It is worth mentioning that an automatic advantage of solving the weighted sum operation (in each iteration) is that a valid inequality/cut can be added to the set of constraints defining $\mathcal{Y}$ which can potentially remove many fractional solutions and improving dual bounds. Specifically, after computing $\bar{\boldsymbol{y}}$ by calling $\text{WSO}(\boldsymbol{\lambda})$, the following cut is valid (due to the optimality of $\bar{\boldsymbol{y}}$ for the weighted sum operation),

$$\sum_{i=1}^{p} \lambda_i y_i \geq \sum_{i=1}^{p} \lambda_i \bar{y}_i.$$

A detailed description of our proposed preprocessing heuristic for computing a good initial global primal bound and adding cuts using the weighted sum operation can be found in Algorithm 2.

Enhancement-II (Exchanging bounds) In our proposed decision space search algorithm, one can view Algorithm 1 as a solution approach for solving a master problem, i.e., a MIL-mMP, and the method proposed by Shao and Ehrgott (2016) as a solution approach for solving its corresponding subproblems, i.e., L-mMPs. Both Algorithm 1 and the method of Shao and Ehrgott (2016) have internal procedures for computing (primal and also dual) bounds when solving their corresponding problems. So, the underlying idea of our second enhancement technique is to effectively employ/exchange the information about these bounds between the master problem and the subproblems. Specifically, during the course of solving a subproblem, it is possible that the method of Shao and Ehrgott (2016) finds some solutions that are feasible for not only the L-mMP (that is trying to solve) but also its corresponding MIL-mMP, i.e., the solutions are luckily not fractional. So, such solutions can be immediately used for updating the global primal/upper bound, i.e., G_{UB}, of Algorithm 1 (if they are better). Moreover, we know that at any time during the course of solving a subproblem, the method of Shao and Ehrgott (2016) maintains a global dual bound for the L-mMP. So, one can immediately terminate solving the L-mMP, if its dual bound is not strictly better/smaller than the global upper bound that Algorithm 1 has already obtained, i.e., G_{UB}.

4. A criterion space search algorithm

In the previous section, we proposed a decision space search algorithm that solves a number of L-mMPs in order to solve a MIL-mMP. To solve each L-mMP, the proposed algorithm solves a number of single-objective linear programs. So, in some sense, the proposed algorithm relies on the power of commercial linear programming solvers for solving a MIL-mMP. In this section, we propose a completely different approach. Specifically, we introduce a new criterion space search algorithm for solving MIL-mMPs. As mentioned in the Introduction, in the field of multi-objective optimization, criterion space search algorithms are referred to solution approaches that work in the space of objective functions. The underlying idea of our proposed approach is to solve a MIL-mMP by solving a number of single-objective mixed integer programs. So, in some sense, the proposed algorithm relies on the power of commercial mixed integer linear programming solvers for solving a MIL-mMP.

In the remaining of this section, we first provide a high-level description of our algorithm, then we explain a detailed description of the algorithm, and finally we explain some implementation issues. However, before doing so, we introduce a key operation that is frequently

used in the algorithm and also a critical proposition that guarantees the correctness of the algorithm. The operation is used for solving a min-max optimization problem and takes a reference point, denoted by $\boldsymbol{y}^l \in \mathbb{R}^p$, as input. The reference point basically defines a *search region*, i.e., $\{\boldsymbol{y} \in \mathbb{R}^p : \boldsymbol{y} \geq \boldsymbol{y}^l\}$, in the criterion space for the min-max operation. For a given reference point $\boldsymbol{y}^l \in \mathbb{R}^p$, the operation attempts to compute a feasible point, denoted by $\boldsymbol{y}^u$, where $\boldsymbol{y}^l \leq \boldsymbol{y}^u$ that has the minimum Chebyshev distance from the reference point by solving the following single-objective mixed integer linear program,

$$\boldsymbol{y}^u \in \arg\min_{\boldsymbol{y} \in \mathcal{Y}} \big\{ z \in \mathbb{R} : z \geq 0 \text{ and } y_i - y_i^l \leq z \ \forall i \in \{1, \ldots, p\} \text{ and } y_i^l \leq y_i \ \forall i \in \{1, \ldots, p\} \big\}. \quad (6)$$

We denote this operation by MIN-MAX($\boldsymbol{y}^l$). Note that if $\boldsymbol{y}^u$ =null then Problem (6) is infeasible. Also, in this paper, we sometimes refer to the point $\boldsymbol{y}^l$ as the lower bound point and to the point $\boldsymbol{y}^u$ as the upper bound point. Next we provide a proposition that builds the foundations of our algorithm. The importance of this proposition will be explained in Section 4.3.

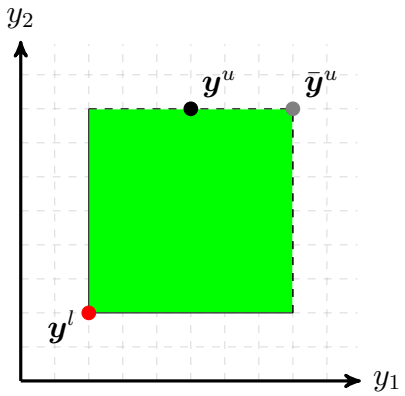
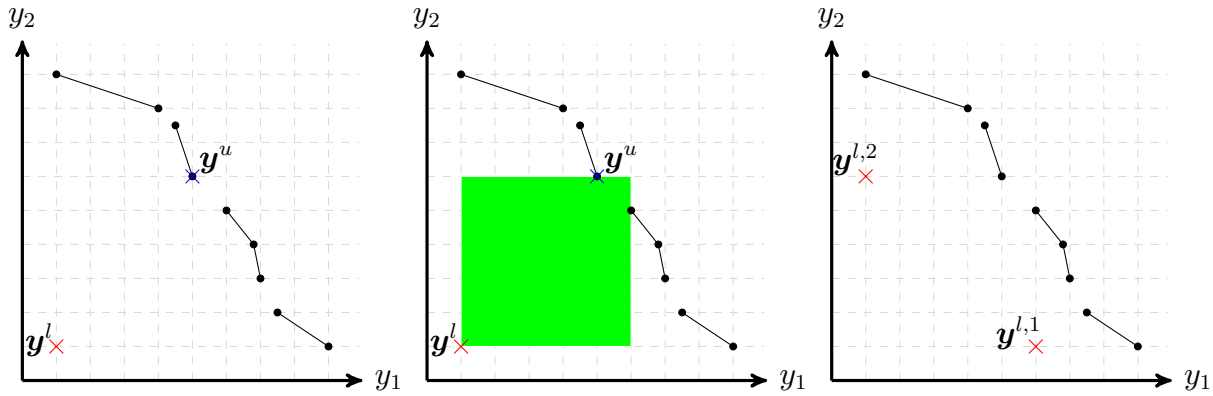


Figure 2 **An illustration of Proposition 2 when** $p = 2$

Informally, the proposition claims that because of using the min-max operation, we can always create a *hypercube* (in the p-dimensional criterion space) containing $\boldsymbol{y}^l$ and $\boldsymbol{y}^u$ with no feasible point in its interior. For example, in Figure 2, an illustration of the points $\boldsymbol{y}^l$ and $\boldsymbol{y}^u$ when $p = 2$ can be found. Observe that in the figure, $y_2^u - y_2^l > y_1^u - y_1^l$. So, let $\bar{y}_1^u := y_1^l + y_2^u - y_2^l$ and $\bar{y}_2^u := y_2^l + y_2^u - y_2^l$. It is evident that the area between $\boldsymbol{y}^l$ and $\bar{\boldsymbol{y}}^u$ is a square, i.e., a hypercube of dimension 2. The proposition claims that there cannot exist any feasible point $\boldsymbol{y}^f$ with $\boldsymbol{y}^f \geq \boldsymbol{y}^l$, i.e., $y_i^f \geq y_i^l$ for $i = 1, \ldots, p$, and $\boldsymbol{y}^f < \bar{\boldsymbol{y}}^u$, i.e., $y_i^f < \bar{y}_i^u$ for $i = 1, \ldots, p$, because of solving the min-max operation to optimality. A formal description of the proposition and its proof are given next.

PROPOSITION 2. *Let* $\boldsymbol{y}^u$ *be an optimal point obtained by* MIN-MAX($\boldsymbol{y}^l$) *and* $z^* := \max\{y_1^u - y_1^l, \ldots, y_p^u - y_p^l\}$ *be the Chebyshev distance of* $\boldsymbol{y}^u$ *from* $\boldsymbol{y}^l$. *Also, let* $\bar{\boldsymbol{y}}^u \in \mathbb{R}^p$ *be a point in the criterion space with* $\bar{y}_i^u := y_i^l + z^*$ *for* $i = 1, \ldots, p$. *There cannot exist any feasible point* $\boldsymbol{y}^f \in \mathcal{Y}$ *such that* $\boldsymbol{y}^f \geq \boldsymbol{y}^l$ *and* $\boldsymbol{y}^f < \bar{\boldsymbol{y}}^u$.

Proof. Suppose not. So, there exists a feasible point $\boldsymbol{y}^f \in \mathcal{Y}$ with $\boldsymbol{y}^f \geq \boldsymbol{y}^l$ and $\boldsymbol{y}^f < \bar{\boldsymbol{y}}^u$. We know that z^* is the Chebyshev distance of $\boldsymbol{y}^u$ from $\boldsymbol{y}^l$ and by construction it should be minimum because of calling MIN-MAX($\boldsymbol{y}^l$). However, since $y_i^f < \bar{y}_i^u = y_i^l + z^*$ for $i = 1, \ldots, p$, we have that $\max\{y_1^f - y_1^l, \ldots, y_1^f - y_1^l\} < z^*$. This immediately implies that $\boldsymbol{y}^u$ cannot be an optimal point of the min-max operation since $\boldsymbol{y}^f$ is feasible and has a strictly smaller Chebyshev distance from $\boldsymbol{y}^l$ (a contradiction). □



(a) Result of the min-max opeartion in the first iteration

(b) Result of Proposition 2 in the first iteration

(c) Branching and creating new lower bound points

Figure 3 An illustration of the key steps of the proposed criterion space search algorithm when $p = 2$

4.1. A high-level description

The proposed criterion space search starts by computing the initial lower bound point. The initial lower bound point is basically the (imaginary) ideal point of the MIL-mMP that we are trying to solve. So, computing the initial lower bound point, denoted by $\boldsymbol{y}^l$, can be done by solving p mixed integer linear programs. Specifically, $y_i^l := \min_{\boldsymbol{y} \in \mathcal{Y}} y_i$ for $i = 1, \ldots, p$. Observe that $\prod_{i=1}^p y_i^l$ is a global dual/lower bound for the optimal objective value of the MIL-mMP. Based on the initial lower bound point, we can now apply the min-max operation to compute an upper bound point $\boldsymbol{y}^u$. Observe that $\prod_{i=1}^p y_i^u$ is a global primal/upper bound for the optimal objective value of the MIL-mMP. An illustration of the nondominated frontier of a MIL-mMP as well as $\boldsymbol{y}^l$ and $\boldsymbol{y}^u$ when $p = 2$ can be found in

Figure 3a. Note that the nondominated frontier of a MIL-mMP can be very complicated since it may not be a convex and/or continuous curve as opposed to the nondominated frontier of a L-mMP. Interested readers may refer to Boland et al. (2015b) to learn more about the nondominated frontier of multi-objective mixed integer linear programs.

Next we can apply Proposition 2 to create an empty-interior hypercube. An illustration of such a hypercube when $p = 2$ is shown in Figure 3b. Afterwards, we can remove the discovered empty region from the search by branching and creating p new lower bound points, denoted by $\boldsymbol{y}^{l,1}, \ldots, \boldsymbol{y}^{l,p}$. In order to do so, we first set $z^* = \max\{y_1^u - y_1^l, \ldots, y_p^u - y_p^l\}$. Now, for each $j \in \{1, \ldots, p\}$, we can define $\boldsymbol{y}^{l,j}$ by setting $y_j^{l,j} = y_j^l + z^*$ and $y_i^{l,j} = y_i^l$ for all $i \in \{1, \ldots, p\} \backslash \{j\}$. An illustration of the new lower bound points when $p = 2$ can be found in Figure 3b.

Observe that each new lower bound point defines a new (but smaller) search region for which the product of the components of its lower bound point is a dual bound. In other words, for each $j \in \{1, \ldots, p\}$, the search region corresponding to $\boldsymbol{y}^{l,j}$ is $\{\boldsymbol{y} \in \mathbb{R}^p : \boldsymbol{y} \geq \boldsymbol{y}^{l,j}\}$ and we know that $\prod_{i=1}^p y_i^{l,j}$ is a dual bound for that search region. So, in each iteration, one can easily update the global dual bound of the MIL-mMP. Specifically, in each iteration, we can first compute the dual bound of each existing (meaning non-yet-explored) search region and then set the global lower bound to the minimum one. Similarly, the global primal bound can be updated in each iteration. Specifically, in each iteration, whenever the algorithm finds an upper bound point, we can first compute the product of its components and then set the global primal bound to it if the current value of global primal bound is (strictly) larger. It is evident that the algorithm can terminate as soon as the global primal bound and global dual bound converge.

4.2. A detailed description

Similar to Algorithm 1, the algorithm maintains a queue of lower bound points, denoted by TREE. The algorithm also maintains a global upper bound, denoted by $\mathrm{G_{UB}}$, and a global lower bound, denoted by $\mathrm{G_{LB}}$. At the beginning, the algorithm sets $\mathrm{G_{UB}} = +\infty$ and $\mathrm{G_{LB}} = -\infty$. The algorithm also computes the ideal point, i.e., $(\min_{\boldsymbol{y} \in \mathcal{Y}} y_1, \ldots, \min_{\boldsymbol{y} \in \mathcal{Y}} y_p)$, and then initializes $\boldsymbol{y}^l$ by setting it equal to the ideal point. After that, the algorithm updates $\mathrm{G_{LB}}$ by setting it equal to $\prod_{i=1}^p y_i^l$ and initializes the queue by $\boldsymbol{y}^l$. The algorithm then explores the queue as long as it is nonempty and $\mathrm{G_{UB}} - \mathrm{G_{LB}} \geq \varepsilon_1$ and $\frac{\mathrm{G_{UB}} - \mathrm{G_{LB}}}{\mathrm{G_{UB}}} \geq \varepsilon_2$,

where $\varepsilon_1, \varepsilon_2 \in (0,1)$ are the user-defined absolute and relative optimality gap tolerances, respectively. Next, we explain how each element of the queue is explored.

In each iteration, the algorithm pops out an element of the queue and denote it by $\boldsymbol{y}^l$. During the course of this research, the best bound strategy, i.e., selecting the node with the minimum dual bound, was found to perform the best for popping out an element. So, in the remaining of this paper, the best bound strategy is set as the default setting of our proposed algorithm. After popping out an element, the algorithm then calls MIN-MAX($\boldsymbol{y}^l$) to compute an upper bound point $\boldsymbol{y}^u$.

If $\boldsymbol{y}^u$=null then the search region corresponding to $\boldsymbol{y}^l$ is infeasible. So, the algorithm simply updates G_{LB} (as discussed in Section 4.1) and starts a new iteration. Otherwise, because $\boldsymbol{y}^u$ is a feasible point, the algorithm checks whether it would be possible to update G_{UB}. Specifically, if $\prod_{i=1}^{p} y_i^u < \mathrm{G_{UB}}$ then the algorithm sets $\mathrm{G_{UB}}$ to $\prod_{i=1}^{p} y_i^u$ and also the best feasible point found, i.e., $\boldsymbol{y}^*$, to $\boldsymbol{y}^u$. Next, the algorithm computes the Chebyshev distance z^* of $\boldsymbol{y}^u$ from $\boldsymbol{y}^l$, i.e., it sets z^* to $\max\{y_1^u - y_1^l, \ldots, y_p^u - y_p^l\}$. Afterwards, the algorithm attempts to create at most p new nodes. To create node $j \in \{1, \ldots, p\}$, the algorithm first generates $\boldsymbol{y}^{l,j}$ based on $\boldsymbol{y}^l$ (as discussed in Section 4.1). The algorithm then adds $\boldsymbol{y}^{l,j}$ to the queue if there is hope to find better feasible points there, i.e., $\mathrm{G_{UB}} - \prod_{i=1}^{p} y_i^{l,j} \geq \varepsilon_1$ and $\frac{\mathrm{G_{UB}} - \prod_{i=1}^{p} y_i^{l,j}}{\mathrm{G_{UB}}} \geq \varepsilon_2$. Finally, the algorithm updates G_{LB} before starting a new iteration.

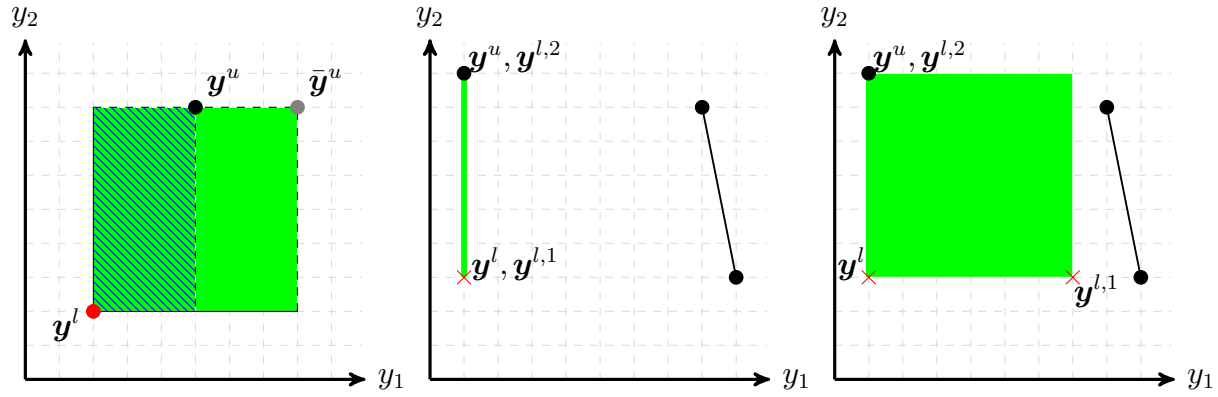
For further details about our proposed criterion space search algorithm, readers can refer to Algorithm 3. We complete this subsection by making one final comment. In multi-objective optimization, one generic issue about criterion space search algorithms (especially for large values of p) is that some nodes in the priority queue could be redundant (see for instance Boland et al. (2017b), Dächert et al. (2017), Kirlik and Sayın (2014)). Hence, in our algorithm, whenever we want to add a node to the priory queue with the lower bound point $\overline{\boldsymbol{y}}^l$, we check whether there is already a node in the tree with the lower bound point $\underline{\boldsymbol{y}}^l$ such that $\underline{y}_i^l \leq \overline{y}_i^l$ for all $i = 1, \ldots, p$. If $\underline{\boldsymbol{y}}^l$ exists then $\overline{\boldsymbol{y}}^l$ is redundant and should not be added to the priority queue. This is because in that case the search region corresponding to $\overline{\boldsymbol{y}}^l$ is a subset of the search region corresponding to $\underline{\boldsymbol{y}}^l$.

4.3. Implementation issues

The main purpose of this section is to explain the importance of Proposition 2 for the correctness of our algorithm. Before doing so, it is worth mentioning that Enhancement-I which is developed for Algorithm 1 (see Section 3.3) can be directly used for Algorithm 3. However, during the course of this research, we found out that the performance

Algorithm 3: A Criterion Space Search Algorithm

Input: A feasible instance of Problem (3)
Queue.create(TREE)
$\mathrm{G_{LB}} \leftarrow -\infty$; $\mathrm{G_{UB}} \leftarrow +\infty$
$\boldsymbol{y}^l \leftarrow (\min_{\boldsymbol{y}\in\mathcal{Y}} y_1, \ldots, \min_{\boldsymbol{y}\in\mathcal{Y}} y_p)$
$\mathrm{G_{LB}} \leftarrow \prod y_i^l$
TREE.*add*$(\boldsymbol{y}^l)$
while *not Queue.empty*(TREE) *&* $\mathrm{G_{UB}} - \mathrm{G_{LB}} \geq \varepsilon_1$ *&* $\frac{\mathrm{G_{UB}}-\mathrm{G_{LB}}}{\mathrm{G_{UB}}} \geq \varepsilon_2$ **do**
    TREE.*PopOut*$(\boldsymbol{y}^l)$
    $\boldsymbol{y}^u \leftarrow$ MIN-MAX$(\boldsymbol{y}^l)$
    if $\boldsymbol{y}^u \neq null$ **then**
        if $\prod_{i=1}^p y_i^u < \mathrm{G_{UB}}$ **then**
            $\mathrm{G_{UB}} \leftarrow \prod_{i=1}^p y_i^u$
            $\boldsymbol{y}^* \leftarrow \boldsymbol{y}^u$
        $z^* \leftarrow \max\{y_1^u - y_1^l, \ldots, y_p^u - y_p^l\}$
        foreach $j \in \{1, \ldots, p\}$ **do**
            $\boldsymbol{y}^{l,j} \leftarrow \boldsymbol{y}^l$; $y_j^{l,j} \leftarrow y_j^l + z^*$
            if $\mathrm{G_{UB}} - \prod_{i=1}^p y_i^{l,j} \geq \varepsilon_1$ *&* $\frac{\mathrm{G_{UB}}-\prod_{i=1}^p y_i^{l,j}}{\mathrm{G_{UB}}} \geq \varepsilon_2$ **then**
                TREE.*add*$(\boldsymbol{y}^{l,j})$
    Update $\mathrm{G_{LB}}$
return $\boldsymbol{y}^*$



(a) An illustration of a box vs a hypercube

(b) Branching based on the area of a box

(c) Branching based on the area of a hypercube

Figure 4 **An illustration of the importance of Proposition 2 when** $p = 2$

of Algorithm 3 does not improve by employing Enhancement-I in practice because (unlike Algorithm 1) the proposed criterion space search algorithm naturally finds feasible solutions/points for a MIL-mMP in each iteration. Also, in theory, one should be able to employ

Enhancement-II for possibly improving the performance of Algorithm 3. However, since each min-max operation in Algorithm 3 will be solved using a commercial solver (and not an open-source solver) in this paper, it would be technically impossible (to the best of our knowledge) to implement Enhancement-II in practice.

We now explain the importance of Proposition 2. Observe that the branching procedure in Algorithm 3 relies on Proposition 2. However, in the literature of criterion space search algorithms for multi-objective optimization problems, this type of branching is not typical. This is because in the literature of multi-objective optimization, there are multiple ways for generating a (nondominated) point and employing a Chebyshev-based operation (similar to the proposed min-max operation) is just one of them (Boland et al. 2015a). By using a min-max operation, it is always possible to identify an empty-interior hypercube in the criterion space according to Proposition 2. However, this observation may not be true for other operations. Specifically, instead of identifying an empty-interior hypercube, other operations can possibly identify empty-interior boxes. However, branching based on boxes rather than hypercubes can be problematic and hence requires further considerations/research.

To see how a box can be created rather than a hypercube, consider Figure 4a in which $p = 2$. Suppose that by calling MIN-MAX($\boldsymbol{y}^l$), we were able to compute $\boldsymbol{y}^u$. By Proposition 2, the area defined by $\boldsymbol{y}^l$ and $\bar{\boldsymbol{y}}^u$ is an empty-interior hypercube. However, one can claim that the box between $\boldsymbol{y}^l$ and $\boldsymbol{y}^u$ is also empty-interior. This implies that if we apply a different operation (than the proposed min-max operation) and finds $\boldsymbol{y}^u$ then we may not be able to claim that $\boldsymbol{y}^l$ and $\bar{\boldsymbol{y}}^u$ is an empty-interior hypercube but at least we can claim that the area between $\boldsymbol{y}^l$ and $\boldsymbol{y}^u$ is an empty-interior box. This is because even if we do not use Proposition 2, we know that there cannot exist any feasible point that strictly dominates $\boldsymbol{y}^u$. So, the question is now why not branching based on the obtained box rather than a hypercube? Of course one can argue that the area that the hypercube will remove is larger. So, it is probably better to use a hypercube. However, we now illustrate a more significant problem that can be created as a result of branching based on a box using Figures 4b and 4c.

Suppose that the nondominated frontier of a MIL-mMP with $p = 2$ consists of a single point and a line segment as shown in Figures 4b. Observe that the result of MIN-MAX($\boldsymbol{y}^l$) is the top endpoint of the nondominated frontier. In this case, the box between MIN-MAX($\boldsymbol{y}^l$)

and $\boldsymbol{y}^u$ is basically a line as shown in Figure 4b. So, if we apply branching based on this line, the new lower bound points will be $\boldsymbol{y}^{l,1} = \boldsymbol{y}^l$ and $\boldsymbol{y}^{l,2} = \boldsymbol{y}^u$. Observe that it is not necessary to add $\boldsymbol{y}^{l,2}$ because $\boldsymbol{y}^{l,2}$ is a feasible point (and Line 17 of Algorithm 3 will take this observation into account). However, $\boldsymbol{y}^{l,1}$ should be added to the queue and this implies that the algorithm falls into an infinite loop since $\boldsymbol{y}^{l,1}$ is exactly $\boldsymbol{y}^l$ (and we have already explored that). However, this issue will never happen if we apply branching based on hypercubes as shown in Figure 4b because $\boldsymbol{y}^{l,1} \neq \boldsymbol{y}^l$.

5. A computational study

In this section, we conduct an extensive computational study and compare the performance of Algorithms 1 and 3 with a generic-purpose solver which is capable of solving MILmMPs, i.e., SCIP 6.0.0. We use Julia 0.6.4 [1] in this section and employ CPLEX 12.8 for solving (single-objective) linear programs and mixed integer linear programs arising during the course of Algorithms 1-3. In addition, all the computational experiments are conducted on a Dell PowerEdge R640 system with two Intel Xeon (Silver 4116) 2.1GHz 12-core processors, 128GB of memory, and RedHat Enterprise Linux 7.4 operating system, and using only a single thread. A time limit of 3600 seconds is imposed for solving each instance and this includes the time of employing any enhancement technique. Our instance generator and codes of this study can be found at `https://github.com/Vahidmhn/MMP_Instance_Generator` and `https://github.com/Vahidmhn/MIMMP.jl`, respectively.

In our computational study, a total of 960 instances are generated and solved. Specifically, for each $p \in \{2, 3, 4\}$, we generate 320 instances. For each $p \in \{2, 3, 4\}$, its corresponding instances are divided into two equal-sized classes denoted by Class-I and Class-II. For the first class, we set $n_c = n_i = n/2$. However, for the second class, we set $n_c = 0$ and $n_i = n$. For convenience, we assume that all integer decision variables are binary. Note that because by our assumptions $\mathcal{X}$ is bounded, in theory, any instance with generic integer decision variables can be transformed to an instance with only binary decision variables with the cost of introducing an additional set of binary variables and constraints. Furthermore, by assuming that all integer decision variables are binary, Class-II contains only pure binary instances that can be easily linearized and solved directly by CPLEX. So, this assumption enables us to directly compare the performance of our methods with CPLEX for linearized

[1] Since the start time of this study, several newer versions of Julia are introduced. However, users can still use our implementation if they employ the configurations described in the links.

instances. For example, consider a MIL-mMP with an objective function of the following form,

$$\min\ (2x_1+3x_2)(x_2+4x_3)=2x_1x_2+8x_1x_3+3x_2^2+12x_2x_3,$$

where $x_1,\ x_2$, and x_3 are binary variables. It is evident that $x_i^2=x_i$ and also any bi-linear term x_ix_j where both x_i and x_j are binary variables can be linearized by adding the following three constraints and introducing a new binary variable q,

$$x_ix_j:=\Big\{q\in\{0,1\}:\ q\le x_i,\ q\le x_j,\ x_i+x_j-1\le q\Big\}.$$

Since the example is in the form of minimization and its coefficients are non-negative, the first two constraints, i.e. $q\le x_i$ and $q\le x_j$, are redundant and can be eliminated. In light of this observation, the linearized objective function is as follows,

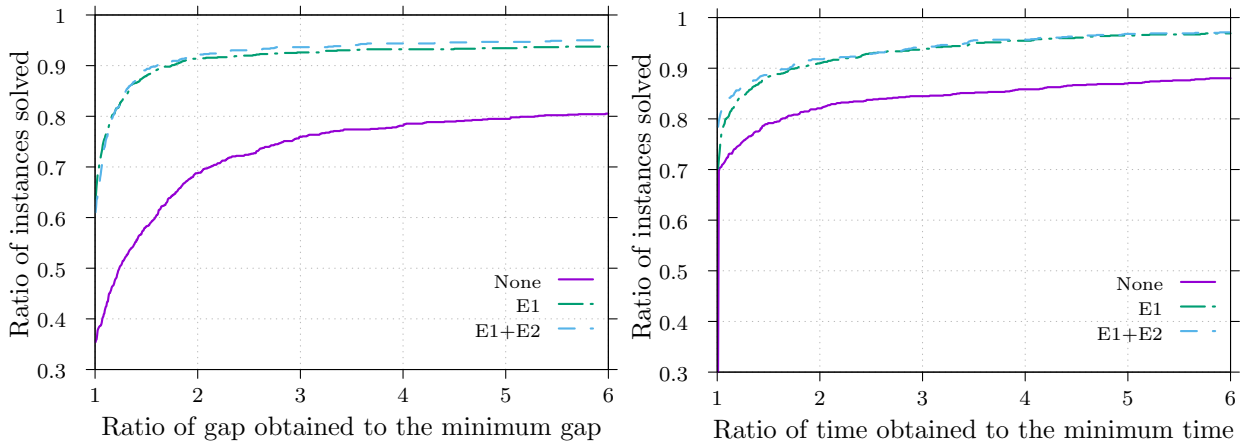
$$\begin{aligned}
\min\ & 2q_1+8q_2+3x_2+12q_3 & \\
\text{s.t. } & x_1+x_2-1\le q_1, & (x_1x_2)\\
& x_1+x_3-1\le q_2, & (x_1x_3)\\
& x_2+x_3-1\le q_3, & (x_2x_3)\\
& q_i\in\{1,0\} & \forall i\in\{1,2,3\}.
\end{aligned}$$

Given the possibility of linearizing instances of Class-II, it is natural to ask whether it is better to use our proposed algorithms to solve them or first linearize such instances and then solve them directly as (single-objective) mixed integer linear programs using CPLEX? We refer to the second approach as *L-CPLEX* in the remaining of this paper and we will explore this question in this computational study. Next, we explain how each class of instances are generated.

Each class of instances in this study contains 16 subclasses with different number of variables n and constraints m. Specifically, each subclass is denoted by $m\times n$ for which we have that $n\in\{400,800,1200,1600\}$ and $m=\alpha\times n$ where $\alpha\in\{1,2,3,4\}$. Each subclass contains 10 random instances and to generate each, the value of the parameters in A, C, and $\boldsymbol{d}$ are randomly generated from the discrete uniform distribution in the interval $[1,10]$. In addition, the sparsity of the matrices A and C are set to 50%, i.e. we set 50% of their elements to zeros. To ensure that $\mathcal{X}$ is bounded (which is an assumption of this study), we impose an upper bound of equal to one for all n decision variables when generating

an instance, i.e., we set $x_i \leq 1$ for all $i = 1, \ldots, n$. Finally, to ensure the feasibility of an instance, the value of b_i is randomly generated from the discrete uniform distribution in the interval $[0, \sum_{j=1}^{n} a_{ij}]$ for each $i \in \{1, \ldots, m\}$.

In this computational study, we frequently use *performance profiles* (Dolan and Moré 2002). A performance profile presents cumulative distribution functions for a set of algorithms being compared with respect to a specific performance metric, i.e., the run time or optimality gap in this study. The run time performance profile for a set of algorithms is constructed by computing for each algorithm and for each instance the ratio of the run time of the algorithm on the instance and the minimum of the run times of all algorithms on the instance. The run time performance profile then shows the ratios on the horizontal axis and, on the vertical axis, for each algorithm, shows the percentage of instances with a ratio that is smaller than or equal to the ratio on the horizontal axis. This implies that values in the upper left-hand corner of the graph indicate the best performance. The optimality gap performance profile can be constructed and interpreted similarly. As an aside, in this study, the optimality gaps and/or the run times are sometimes close to zero and this can create problems when computing ratios in the performance profiles. To resolve this issue, when creating performance profiles, we first add 0.01% to all reported optimality gaps and 0.01 second to all reported run times.

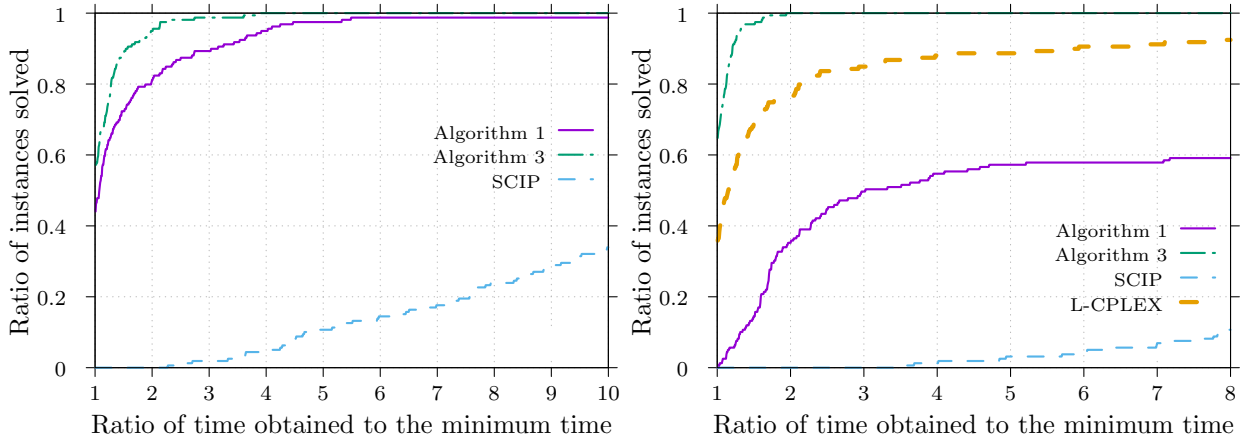


(a) Optimality gap (b) Run time

Figure 5 Performance profiles of the proposed enhancement techniques on Algorithm 1 over all instances

Finally, as mentioned in Section 3, Algorithm 1 can be potentially improved by employing Enhancement-I (E1) and Enhancement-II (E2). So, before we compare the performance

of different algorithms, we check whether activating the proposed enhancement techniques can be helpful. Figure 5 illustrates the run time and optimality gap performance profiles of Algorithm 1 under three different scenarios: no enhancement is active (Scenario 1), only E1 is active (Scenario 2), both E1 and E2 are active (Scenario 3), on all our 960 instances. We observe that both the run time and optimality gap performance profiles under Scenario 3 are slightly better than Scenario 2 but they are significantly better than Scenario 1. Specifically, by comparing Scenarios 1 and 3, we observe that around 10% of instances are solved 6 times faster. Also, the optimality gap of around 15% of instances are at least 6 times smaller. Hence, in the remaining of this section, whenever we refer to Algorithm 1, we assume that both enhancements are active.



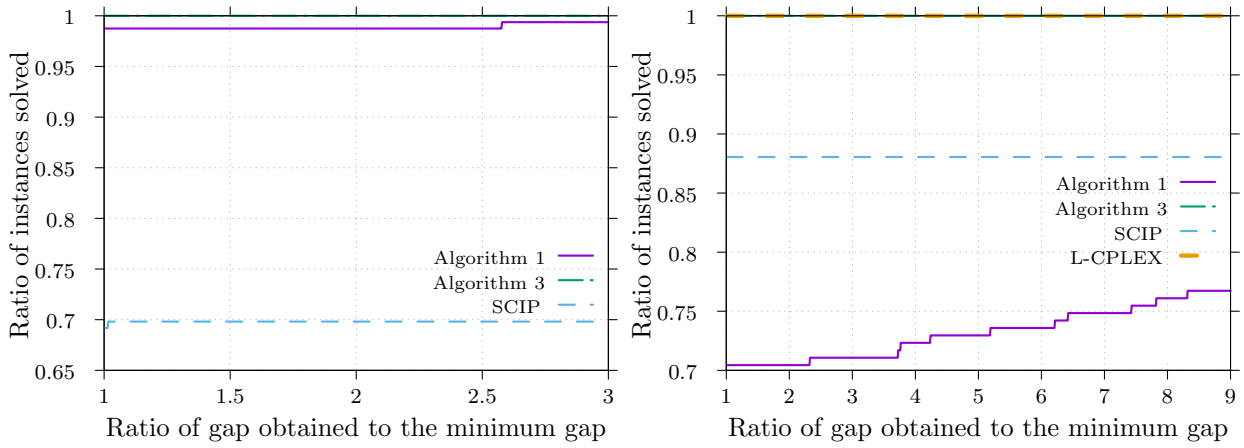
(a) Class-I (b) Class-II

Figure 6 **Run time performance profiles when $p = 2$**

5.1. Two objectives ($p = 2$)

In this section, we compare the performance of different methods including Algorithm 1, Algorithm 3, SCIP, and L-CPLEX, on solving instances with $p = 2$. The run time performance profiles of the different methods on instances of Class-I and Class-II can be found in Figure 6. Also, the optimality gap performance profiles of different methods on instances of Class-I and Class-II can be found in Figure 7.

Note that L-CPLEX is only tested on instances of Class-II since they do not have any continuous decision variables. Observe that for both Class-I and Class-II, Algorithm 3 has the best performance and SCIP has the worst performance. For Class-I, Algorithm 3



(a) Class-I (b) Class-II

Figure 7 **Optimality gap performance profiles when** $p = 2$

has been able to solve around 10% and 98% of instances at least 3 times faster than Algorithm 1 and SCIP, respectively. For Class-II, Algorithm 3 has been able to solve around 16%, 50%, and 99% of instances at least 3 times faster than L-CPLEX, Algorithm 1, and SCIP, respectively. So, the second best method for solving instances of Class-II has been L-CPLEX. However, the difference between Algorithm 3 and L-CPLEX is significant in Class-II and this is highlighted by this observation that around 8% of instances are solved at least 8 times faster using Algorithm 3. In terms of the optimality gap, a similar pattern can be observed. Specifically, Algorithm 3 has been able to solve all instances to optimality within the imposed time limit. However, Algorithm 1 has performed poorly on Class-II but has been able to solve almost all instances of Class-I within the imposed time limit. This is mainly because, by construction, Class-I contains significantly less number of integer decision variables compared to Class-II. Therefore Algorithm 1, which is a decision space search algorithm, struggles to solve such instances because the size of its search tree depends highly on the number of integer decision variables. Finally, we also observe that, for around 30% of instances, Algorithm 3 has reached to an optimality gap which is at least 3 times smaller than SCIP on Class-I and Algorithm 1 on Class-II.

Table 1 provides further details about the performance of Algorithms 1 and 3 on instances with $p = 2$. Columns labeled 'Time (s)' show the solution time in seconds, columns labeled 'Gap (%)' show the optimality gap, and columns labeled 'Solved (#)' show the number of instances solved to optimality. Numbers in this table, except those in Columns labeled

'Solved (#)', are averages over 10 instances. The table clearly shows that Algorithm 3 outperforms Algorithm 1 significantly. Observe that the largest subclass of instances are solved to optimality in around 50.72 seconds and 259.16 seconds using Algorithm 3 under Class-I and Class-II, respectively. However, Algorithms 1 was not able to solve 1 and 6 instances of the largest subclass of instances to optimality within the imposed time limit under Class-I and Class-II, respectively.

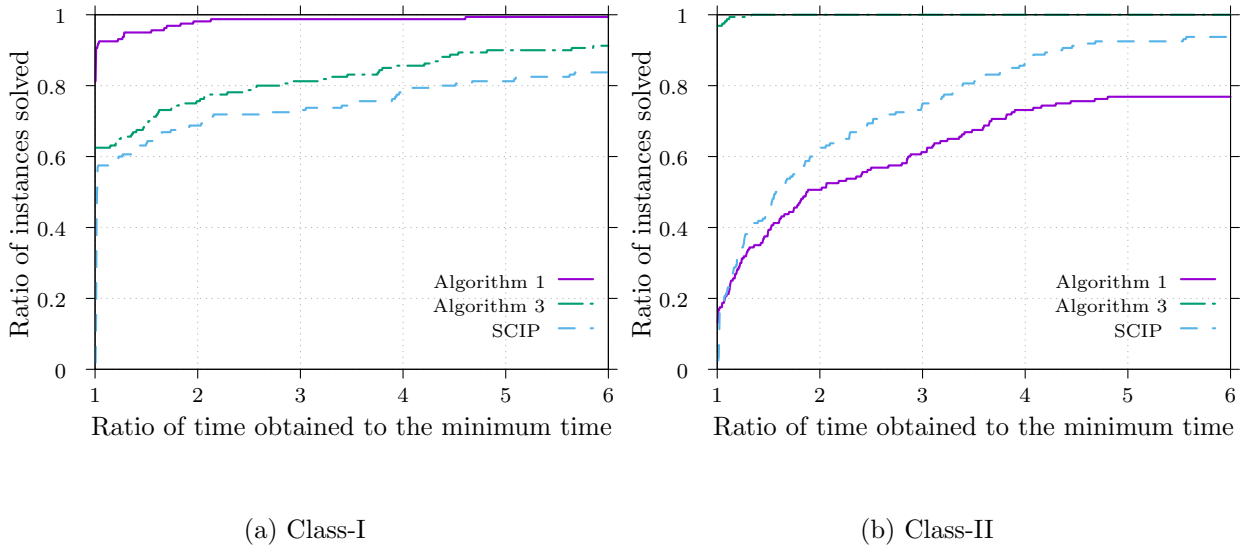
Table 1 **Performance comparison of Algorithms 1 and 3 on instances with $p = 2$**

Subclass	Class-I						Class-II					
	Algorithm 1			Algorithm 3			Algorithm 1			Algorithm 3		
	Time (s)	Gap (%)	Solved (#)	Time (s)	Gap (%)	Solved (#)	Time (s)	Gap (%)	Solved (#)	Time (s)	Gap (%)	Solved (#)
400 × 400	5.43	0.00	10	3.07	0	10	5.53	0.00	10	1.06	0	10
800 × 400	20.17	0.00	10	7.00	0	10	501.63	0.03	9	2.26	0	10
1200 × 400	47.15	0.00	10	14.46	0	10	1674.59	0.08	6	3.56	0	10
1600 × 400	20.25	0.00	10	14.90	0	10	2199.49	1.09	4	7.08	0	10
800 × 800	10.96	0.00	10	6.57	0	10	10.14	0.00	10	1.15	0	10
1600 × 800	20.72	0.00	10	32.72	0	10	796.25	1.45	8	4.40	0	10
2400 × 800	147.65	0.00	10	40.35	0	10	733.05	0.54	8	7.05	0	10
3200 × 800	334.57	0.00	10	98.65	0	10	2224.25	3.37	4	16.63	0	10
1200 × 1200	15.17	0.00	10	15.83	0	10	15.95	0.00	10	3.30	0	10
2400 × 1200	105.38	0.00	10	50.35	0	10	880.78	0.47	8	8.38	0	10
3600 × 1200	506.98	0.18	9	68.54	0	10	1464.86	1.41	7	19.94	0	10
4800 × 1200	132.27	0.00	10	145.50	0	10	2888.05	1.43	2	46.29	0	10
1600 × 1600	20.16	0.00	10	17.16	0	10	24.60	0.00	10	5.69	0	10
3200 × 1600	74.18	0.00	10	102.61	0	10	860.45	0.22	9	19.83	0	10
4800 × 1600	80.60	0.00	10	129.56	0	10	3077.95	5.25	4	38.06	0	10
6400 × 1600	695.01	0.00	9	259.16	0	10	2621.50	10.00	4	50.72	0	10

5.2. Three objectives ($p = 3$)

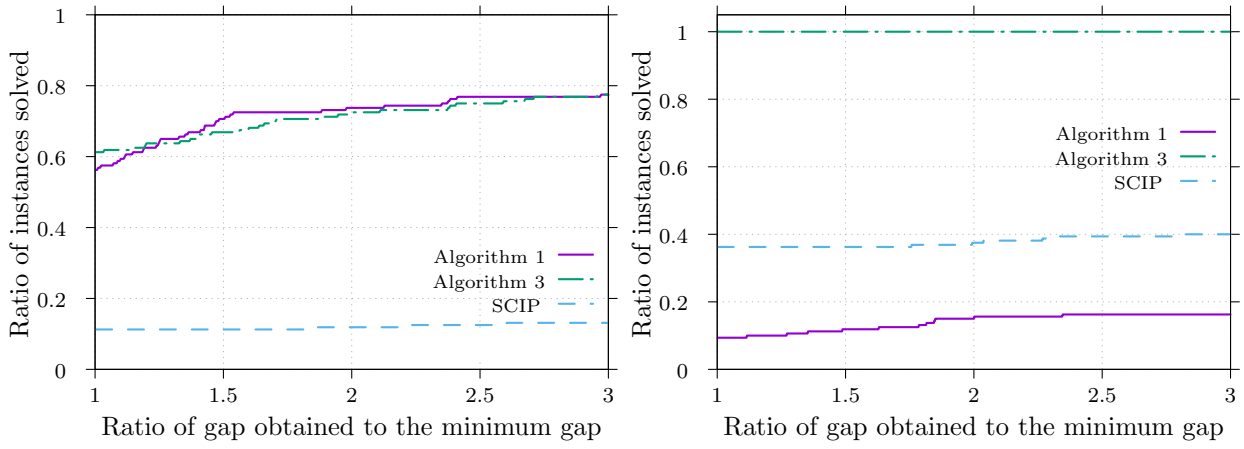
In this section, we compare the performance of different methods including Algorithm 1, Algorithm 3, and SCIP, on solving instances with $p = 3$. Note that, in the remaining of this paper, we do not provide any further result for L-CPLEX because for $p > 2$ many constraints and decision variables need to be added for linearizion and CPLEX will struggle to solve such linearized instances. With this in mind, the run time performance profiles of

the different methods on instances of Class-I and Class-II can be found in Figure 8. Also, the optimality gap performance profiles of different methods on instances of Class-I and Class-II can be found in Figure 9.



(a) Class-I (b) Class-II

Figure 8 **Run time performance profiles when $p = 3$**



(a) Class-I (b) Class-II

Figure 9 **Optimality gap performance profiles when $p = 3$**

One interesting observation is that, unlike instances with $p = 2$, Algorithm 1 outperforms Algorithm 3 for Class-I. Specifically, Algorithm 1 has been able to solve around 20% and 30% of instances at least 3 times faster than Algorithm 3 and SCIP, respectively. This is not surprising because of two reasons. (1) Class-I involves less integer decision

variables and hence a decision space search algorithm can be expected to perform well for such instances. (2) Algorithm 3 is a criterion space search algorithm and hence as p increases, more nodes need to be added after exploring each node of the search tree. So, the global dual bound is expected to improve at slower rate during the search. For Class II, however, Algorithm 3 has the best performance and Algorithm 1 has the worst performance. Specifically, Algorithm 3 has been able to solve around 40% and 30% of instances at least 3 times faster than Algorithm 1 and SCIP, respectively. In terms of the optimality gap, for Class-I, Algorithms 1 and 3 have similar performance. However, for around 70% of instances of Class-I, Algorithm 3 has reached to an optimality gap which is at least 3 times smaller than SCIP. For Class II, Algorithm 1 has performed poorly in terms of the optimality gap. Specifically, we observe that, for around 80% of instances in Class-II, Algorithm 3 has reached to an optimality gap which is at least 3 times smaller than Algorithm 1.

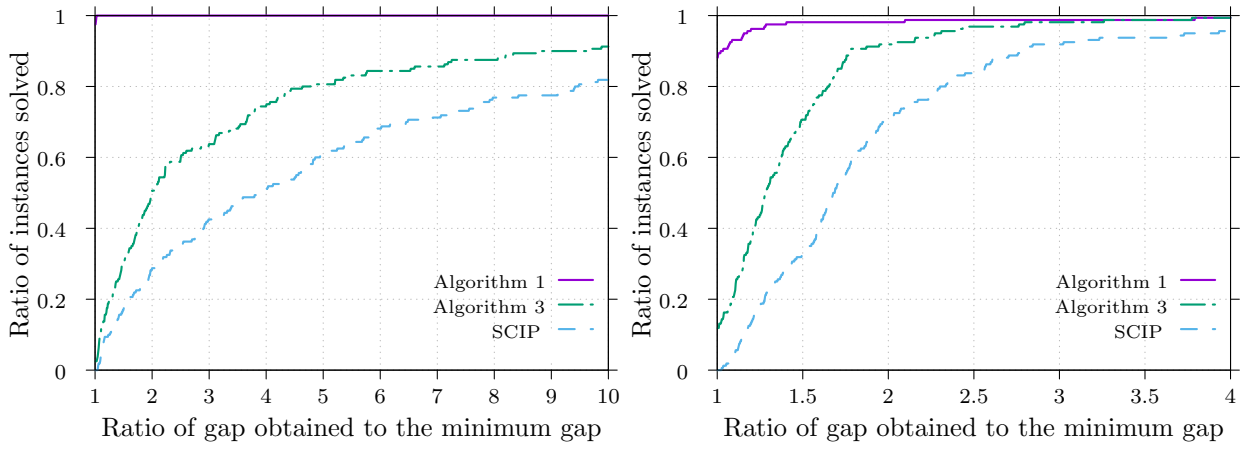
Table 2 Performance comparison of Algorithms 1 and 3 on instances with $p = 3$

Subclass	Class-I						Class-II					
	Algorithm 1			Algorithm 3			Algorithm 1			Algorithm 3		
	Time (s)	Gap (%)	Solved (#)	Time (s)	Gap (%)	Solved (#)	Time (s)	Gap (%)	Solved (#)	Time (s)	Gap (%)	Solved (#)
400 × 400	1762.16	3.01	6	2184.25	0.18	5	3135.94	9.73	2	98.45	0.00	10
800 × 400	1925.29	2.99	5	3334.84	0.34	2	3600.00	18.25	0	377.61	0.00	10
1200 × 400	1956.08	10.02	5	3405.11	0.19	2	3600.00	14.38	0	682.83	0.00	10
1600 × 400	3000.39	4.99	2	3357.12	0.23	1	3600.00	16.55	0	1403.17	0.30	9
800 × 800	1234.85	1.52	7	1991.53	0.04	6	2882.26	5.65	2	103.00	0.00	10
1600 × 800	2229.45	3.27	4	3295.56	2.80	2	3600.00	16.24	0	680.56	0.00	10
2400 × 800	3258.70	5.65	1	3600.00	2.26	0	3600.00	24.81	0	1098.76	0.00	10
3200 × 800	3204.88	5.24	2	3600.00	3.46	0	3600.00	23.62	0	2226.01	0.34	8
1200 × 1200	1115.15	0.75	7	1612.77	0.02	8	2618.78	5.86	4	231.66	0.00	10
2400 × 1200	2125.83	4.38	6	3385.22	10.12	1	3600.00	15.21	0	1130.86	0.00	10
3600 × 1200	3198.68	4.89	2	3600.00	5.37	0	3600.00	29.85	0	2907.52	3.05	5
4800 × 1200	3331.31	5.90	1	3600.00	12.24	0	3600.00	24.61	0	3145.63	5.61	4
1600 × 1600	504.71	0.35	9	1768.84	0.76	7	2634.85	3.41	4	241.15	0.00	10
3200 × 1600	2954.51	8.21	3	3250.68	8.63	1	3600.00	14.94	0	1354.44	0.00	10
4800 × 1600	2799.84	6.64	4	3156.85	9.78	2	3600.00	18.46	0	2839.52	2.92	5
6400 × 1600	3600.00	9.54	0	3600.00	21.70	0	3600.00	26.24	0	3320.17	7.10	4

Table 2 provides further details about the performance of Algorithms 1 and 3 on instances with $p = 3$. Observe that the solution times have increased dramatically for instances with $p = 3$ compared to instances with $p = 2$. It is evident that Algorithm 1 has been able to solve significantly more instances to optimality for Class-I. However, for Class-II, Algorithm 3 has solved significantly more instances to optimality. The average optimality gap reported for each subclass indicates a similar pattern. Specifically, for the largest subclass of instances, we observe that the average optimality of Algorithm 3 is more than 2.2 times larger than Algorithm 1 under Class-I but it is more than 3.6 times smaller under Class-II.

5.3. Four objectives ($p = 4$)

In this section, we compare the performance of different methods including Algorithm 1, Algorithm 3, and SCIP, on solving instances with $p = 4$. Unlike the previous cases, i.e., $p = 2$ and $p = 3$, we do not provide any run time performance profiles because almost all instances are not solved to optimality by any of the solution methods within the imposed time limit. The optimality gap performance profiles of different methods on instances of Class-I and Class-II can be found in Figure 10.



(a) Class-I (b) Class-II

Figure 10 **Optimality gap performance profiles when $p = 4$**

From the figure, we observe that Algorithm 1 has completely outperformed Algorithm 3 in terms of the optimality gap. Specifically, for around 20% and 40% of instances of Class-I, Algorithm 1 has reached to an optimality gap which is at least 5 times better than those obtained by Algorithm 3 and SCIP, respectively. Similarly, for around 30% and

70% of instances of Class-II, Algorithm 1 has reached to an optimality gap which is at least 1.5 times better than those obtained by Algorithm 3 and SCIP, respectively. Table 3 provides further details about the performance of Algorithms 1 and 3 on instances with $p = 4$. Observe that very few instances are solved to optimality by the methods within the imposed time limit. The average optimality gap reported for each subclass indicates that Algorithm 1 has completely outperformed Algorithm 3. This is another evidence that for large values of p, the proposed decision space search algorithm is a better choice.

Table 3 Performance comparison of Algorithms 1 and 3 on instances with $p = 4$

Subclass	Class-I						Class-II					
	Algorithm 1			Algorithm 3			Algorithm 1			Algorithm 3		
	Time (s)	Gap (%)	Solved (#)	Time (s)	Gap (%)	Solved (#)	Time (s)	Gap (%)	Solved (#)	Time (s)	Gap (%)	Solved (#)
400 × 400	3282.43	15.88	1	3600.00	59.25	0	3600.00	24.73	0	3600.00	36.57	0
800 × 400	3600.00	12.43	0	3600.00	30.31	0	3600.00	31.26	0	3600.00	56.83	0
1200 × 400	3600.00	12.47	0	3600.00	38.53	0	3600.00	27.60	0	3600.00	63.95	0
1600 × 400	3600.00	10.43	0	3600.00	36.79	0	3600.00	36.72	0	3600.00	67.02	0
800 × 800	3273.97	9.69	1	3600.00	55.89	0	3600.00	23.57	0	3504.52	40.83	1
1600 × 800	3087.01	18.51	2	3600.00	65.06	0	3600.00	36.15	0	3600.00	72.58	0
2400 × 800	3600.00	22.51	0	3600.00	63.75	0	3600.00	36.73	0	3600.00	73.35	0
3200 × 800	3600.00	11.90	0	3600.00	68.32	0	3600.00	30.67	0	3600.00	76.33	0
1200 × 1200	3600.00	12.18	0	3600.00	47.08	0	3600.00	32.21	0	3600.00	54.36	0
2400 × 1200	3600.00	18.90	0	3600.00	69.25	0	3600.00	45.16	0	3600.00	69.40	0
3600 × 1200	3600.00	18.00	0	3600.00	81.08	0	3600.00	50.78	0	3600.00	78.44	0
4800 × 1200	3600.00	40.70	0	3600.00	70.59	0	3600.00	24.53	0	3600.00	85.33	0
1600 × 1600	3323.96	6.67	1	3600.00	50.39	0	3600.00	35.52	0	3600.00	47.60	0
3200 × 1600	3567.38	12.69	1	3600.00	74.71	0	3600.00	49.32	0	3600.00	68.65	0
4800 × 1600	3600.00	55.16	0	3600.00	84.29	0	3600.00	47.78	0	3600.00	76.75	0
6400 × 1600	3600.00	49.08	0	3600.00	82.58	0	3600.00	61.08	0	3600.00	84.45	0

6. Conclusions

We studied a class of single-objective non-convex non-linear (mixed) integer optimization problems, i.e., MIL-mMPs, with applications in different fields of study, e.g., natural resource management. We showed that because the objective function of a MIL-mMP is a multiplication of p non-negative linear functions, a MIL-mMP can be viewed as the problem of optimization over the efficient set of a multi-objective mixed integer linear

program with p objective functions. Consequently, we developed two novel multi-objective optimization based algorithms for solving MIL-mMPs. The first algorithm is a decision space search algorithm that relies on the power of (single-objective) linear programming solvers for solving a MIL-mMP. The second algorithm, however, is a criterion space search algorithm that relies on the power of (single-objective) mixed integer linear programming solvers for solving a MIL-mMP.

We developed several enhancement techniques for the proposed algorithms and tested the performance of our algorithms on 960 random instances against the non-linear optimization solver of SCIP. Our numerical results showed that SCIP is significantly outperformed by our proposed algorithms. We also showed that commercial mixed integer programming solvers struggle to directly solve instances of MIL-mMP when their objective functions are linearized in advance. The results showed that when $p = 2$ the proposed criterion space search algorithm is the best solution method. However, as p increases, the proposed criterion space search algorithm starts to struggle because the dimension of the criterion space increases. Consequently, for $p = 4$, we observed that the proposed decision space search algorithm outperforms the proposed criterion space search algorithm. Similarly, we also observed that as the number of integer decision variables increases, the proposed decision space search algorithm starts to struggle because the dimension of the decision space increases. Consequently, for $p = 3$, we observed that the decision space search algorithm was the best choice for our mixed integer instances because they involved less number of integer decision variables (compared to our pure integer instances). However, for pure integer instances with $p = 3$, the criterion space search algorithm was the best choice because they involved more integer decision variables (compared to our mixed integer instances).

We hope that the simplicity, versatility, and performance of our proposed algorithms encourage practitioners/researchers to consider developing/employing multi-objective optimization based algorithms for solving certain single-objective optimization problems in particular multiplicative programs. There are several future research directions that can be considered for this study. One direction could be exploring whether the proposed algorithms can be combined together through an effective parallelization framework to integrate their advantageous. Another research direction is to explore how the proposed methods can be customized effectively for solving the so-called 'generalized' MIL-mMPs in which the only difference is that each term in the objective function may have a positive power.

References

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

Benson H, Boger G (1997) Multiplicative programming problems: analysis and efficient point search heuristic. *Journal of Optimization Theory and Applications* 94(2):487–510.

Benson H, Boger G (2000) Outcome-space cutting-plane algorithm for linear multiplicative programming. *Journal of Optimization Theory and Applications* 104(2):301–322.

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

Benson HP (1999) An outcome space branch and bound-outer approximation algorithm for convex multiplicative programming. *Journal of Global Optimization* 15(4):315–342.

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

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

Boland N, Charkhgard H, Savelsbergh M (2017a) A new method for optimizing a linear function over the efficient set of a multiobjective integer program. *European Journal of Operational Research* 260(3):904 – 919.

Boland N, Charkhgard H, Savelsbergh M (2017b) The quadrant shrinking method: A simple and efficient algorithm for solving tri-objective integer programs. *European Journal of Operational Research* 260(3):873 – 885.

Charkhgard H, Savelsbergh M, Talebian M (2018) A linear programming based algorithm to solve a class of optimization problems with a multi-linear objective function and affine constraints. *Computers & Operations Research* 89:17 – 30, `https://doi.org/10.1016/j.cor.2017.07.015`.

Chen PC, Hansen P, Jaumard B (1991) On-line and off-line vertex enumeration by adjacency lists. *Operations Research Letters* 10(7):403 – 409, ISSN 0167-6377, URL `http://dx.doi.org/https://doi.org/10.1016/0167-6377(91)90042-N`.

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

Dächert K, Klamroth K, Lacour R, Vanderpooten D (2017) Efficient computation of the search region in multi-objective optimization. *European Journal of Operational Research* 260(3):841 – 855.

Dolan ED, Moré JJ (2002) Benchmarking optimization software with performance profiles. *Mathematical Programming* 91(2):201–213.

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

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

Eusébio A, Figueira J, Ehrgott M (2014) On finding representative non-dominated points for bi-objective integer network flow problems. *Computers & Operations Research* 48:1 – 10.

Gao Y, Wu G, Ma W (2010) A new global optimization approach for convex multiplicative programming. *Applied Mathematics and Computation* 216(4):1206–1218.

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

Kim NTB, Le Trang NT, Yen TTH (2007) Outcome-space outer approximation algorithm for linear multiplicative programming. *East-West Journal of Mathematics* 9(1).

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

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

Konno H, Kuno T (1992) Linear multiplicative programming. *Mathematical Programming* 56(1-3):51–64.

Kuno T (2001) A finite branch-and-bound algorithm for linear multiplicative programming. *Computational Optimization and Applications* 20(2):119–135.

Kuno T, Yajima Y, Konno H (1993) An outer approximation method for minimizing the product of several convex functions on a convex set. *Journal of Global optimization* 3(3):325–335.

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

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

Oliveira RM, Ferreira PA (2008) A convex analysis approach for convex multiplicative programming. *Journal of Global Optimization* 41(4):579–592.

Özlen M, Azizoğlu M (2009) Multi-objective integer programming: A general approach for generating all non-dominated solutions. *European Journal of Operational Research* 199:25–35.

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

Pal A, Charkhgard H (2019) FPBH: A feasibility pump based heuristic for multi-objective mixed integer linear programming. *Computers & Operations Research* 112:104760.

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

Saghand PG, Charkhgard H, Kwon C (2019) A branch-and-bound algorithm for a class of mixed integer linear maximum multiplicative programs: A bi-objective optimization approach. *Computers & Operations Research* 101:263–274.

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

Shao L, Ehrgott M (2014) An objective space cut and bound algorithm for convex multiplicative programmes. *Journal of Global Optimization* 58(4):711–728.

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

Sierra Altamiranda A, Charkhgard H (2019) A new exact algorithm to optimize a linear function over the set of efficient solutions for biobjective mixed integer linear programs. *INFORMS Journal on Computing* 31(4):823–840.

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

Van Thoai N (1991) A global optimization approach for solving the convex multiplicative programming problem. *Journal of Global Optimization* 1(4):341–357.

Acknowledgments

This work was supported by the National Science Foundation under Grant No 1849627.