A Criterion Space Search Algorithm for Mixed Integer Linear Maximum Multiplicative Programs: A Multi-objective Optimization Approach

Payman Ghasemi Saghand
payman@mail.usf.edu; Industrial and Management Systems Engineering, University of South Florida, Tampa, Florida 33620, USA

Hadi Charkhgard
hcharkhgard@usf.edu;Industrial and Management Systems Engineering, University of South Florida, Tampa, Florida 33620, USA

We study a class of mixed integer optimization problems with linear constraints and a multi-linear objective function, the so-called mixed integer linear maximum multiplicative programs (MIL-MMPs). Such a problem can be transformed into a Second-order Cone Program (SOCP) and can be solved effectively by a commercial solver such as CPLEX. However, MIL-MMPs can also be viewed as special cases of the problem of optimization over the set of efficient solutions in multi-objective optimization. By using this observation, we develop a criterion space search algorithm for solving any MIL-MMP. An extensive computational study on around 2000 instances illustrates that the proposed algorithm significantly outperforms not only the CPLEX Mixed Integer SOCP solver but also a state-of-the-art algorithm which is capable of solving special cases of MIL-MMPs. Moreover, the computational study illustrates that even if we linearize the objective function and solve the linearized problem by CPLEX, the proposed algorithm still performs significantly better.

Key words: Mixed integer maximum multiplicative programming, Multi-objective optimization, Optimization over the efficient set, Criterion space search algorithm

1. Introduction

Most real-life optimization problems involve multiple (conflicting) objective functions and they can be stated as follows:

$$\max_{\boldsymbol{x}\in\mathcal{X}} \{y_1(\boldsymbol{x}),\ldots,y_p(\boldsymbol{x})\}, \tag{1}$$

where $\mathcal{X} \subseteq \mathbb{R}^n$ represents the set of feasible solutions of the problem, and $y_1(\boldsymbol{x}), \ldots, y_p(\boldsymbol{x})$ are p objective functions. Overall, the conflict between objective functions makes it almost impossible to find a feasible/ideal solution that optimizes all objectives simultaneously. Hence, in multi-objective optimization, the focus is on computing *efficient* solutions, i.e., solutions in which it is impossible to improve the value of one objective without deteriorating the values of other objectives. So, it is not surprising that in recent years, a significant amount of effort has been conducted on computing the set of efficient solutions of multi-objective optimization problems entirely or partially, see for instance Karasakal and Köksalan (2009), Sayın (2003), Masin and Bukchin (2008), Boland et al. (2017b), Dächert and Klamroth (2014), Bozkurt et al. (2010), Kirlik and Sayın (2014), Lokman and Köksalan (2013), Özlen et al. (2013), Özpeynirci and Köksalan (2010), Przybylski and Gandibleux (2017), Soylu and Yıldız (2016), Ehrgott and Gandibleux (2007), and Eusébio et al. (2014).

Although computing the efficient solutions of a multi-objective optimization problem helps us understand the trade-offs between the objective functions, some researchers argue that presenting too many efficient solutions can confuse a decision maker and may make selecting a preferred solution almost impossible (Jorge 2009, Boland et al. 2017a). An approach that can resolve this issue is finding a preferred solution among the set of efficient solutions directly (Sayın 2000, Jorge 2009). This approach is known as *optimizing over the efficient set*, which is a global optimization problem (Benson 1984). Let $\mathcal{X}_E$ be the set of efficient solutions of Problem (1). Also, let $g(\boldsymbol{x})$ be a function representing how decision makers will choose their preferred solutions from $\mathcal{X}_E$. The problem of optimizing over the efficient set can be stated as follows:

$$\max_{\boldsymbol{x} \in \mathcal{X}_E} g(\boldsymbol{x}).$$

A specific subclass of such optimization problems is *maximum multiplicative programs (MMPs)*, i.e., problems of the form:

$$\max_{\boldsymbol{x} \in \mathcal{X}} \prod_{i=1}^{p} y_i(\boldsymbol{x}),$$

where $y_i(\boldsymbol{x}) \geq 0$ for all $\boldsymbol{x} \in \mathcal{X}$ and $i = 1, \ldots, p$. It is assumed that the optimal objective value of an MMP is strictly positive and $\mathcal{X}$ is bounded. At first glance, an MMP does not look like a special case of the problem of optimizing over the efficient set. However, it is known (see for example Nash 1950) that an MMP is equivalent to the following problem:

$$\max_{\boldsymbol{x}\in\mathcal{X}_E} \prod_{i=1}^{p} y_i(\boldsymbol{x}).$$

where $\mathcal{X}_E$ is the set of efficient solutions of Problem (1). Hence, by letting $g(\boldsymbol{x}) = \prod_{i=1}^{p} y_i(\boldsymbol{x})$, we observe that an MMP is a special case of the problem of optimization over the efficient set.

The focus of this study is on Mixed Integer *Linear* MMPs (MIL-MMPs), those in which $\mathcal{X}$ is defined by a set of linear constraints, $y_i(\boldsymbol{x})$ is linear for all $i = 1, \cdots, p$, and the vector of decision variables, i.e. $\boldsymbol{x}$, can contain continuous as well as integer decision variables. MIL-MMPs have significant number of applications in different fields of study such as game theory, conservation planning, statistics, and system reliability (Charkhgard et al. 2018). Some of which are explained in Section 2.

Due to the importance of MIL-MMPs, several techniques have been developed to solve *continuous* MIL-MMPs, i.e., those with no integer decision variables. Evidently, a convex programming solver, for example one that uses an interior point method, can solve a continuous MIL-MMPs to optimality in polynomial time (Grötschel et al. 1988). Also, since we assume that the optimal objective value of an MMP is strictly positive, one can use the log-transformation of the objective function, i.e., $\sum_{i=1}^{p} \log z_i(\boldsymbol{x})$, and solve a continuous MIL-MMPs even faster in practice using a convex programming solver (Charkhgard et al. 2018). However, one of the fastest approach (in practice) to solve a continuous MMP is to transform the problem into a Second-Order Cone Program (SOCP) using the technique developed by Ben-Tal and Nemirovski (2001) and solve such a SOCP using commercial solvers such as CPLEX (Charkhgard et al. 2018). In Section 3, we review how the transformation of a MIL-MMP to a SOCP can be done. Since linear programming solvers are significantly faster than convex programming solvers, some authors (see for instance Vazirani (2012)) showed that it is possible to develop a linear programming based algorithm for solving continuous MIL-MMPs. In the same direction, Charkhgard et al. (2018) used the observation that a continuous MIL-MMP is a special case of the problem of optimizing a linear function over the set of efficient solutions of multi-objective linear programs. So, they developed a linear programming based algorithm that can solve continuous MIL-MMPs when $p = 2$. They numerically showed that their algorithm outperforms CPLEX SOCP solver by a factor of up to three for large-sized instances.

Note that continuous MIL-MMPs appear to be closely related to continuous linear *minimum* multiplicative programs (see for instance Gao et al. (2006), Ryoo and Sahinidis (2003), Shao and Ehrgott (2014); and Shao and Ehrgott (2016)). Specifically, by changing the objective function of a continuous MIL-MMP from *max* to *min* a continuous linear minimum multiplicative program is obtained. However, it is known that a linear minimum multiplicative program is NP-hard (Shao and Ehrgott 2016). Therefore, because of this significant difference, we do not consider this class of optimization problems in this study.

As mentioned above, there are several studies about solving continuous MIL-MMPs. However, the only existing study (to the best of our knowledge) for solving instances with integer decision variables is our recent paper (see Saghand et al. (2019)). Note that SOCP solver of CPLEX can solve any MIL-MMPs (regardless of whether there are some integer variables in the model or not). However, in our recent work, we showed that the algorithm proposed by Charkhgard et al. (2018) (for continuous MIL-MMPs with $p = 2$) can be used in an effective branch-and-bound framework for solving any MIL-MMPs with $p = 2$. The computational study conducted by Saghand et al. (2019) showed that the proposed branch-and-bound algorithm outperforms CPLEX SOCP by a factor of around 2 in most instances (with $p = 2$).

In light of the above, the main contribution of our research is developing an algorithm that can solve any MIL-MMPs with any $p \geq 2$. The proposed algorithm is the first *criterion space search algorithm* for solving MIL-MMPs. In the literature of multi-objective optimization, an algorithm is called a criterion space search algorithm if it solves a multi-objective optimization problem by working on the space of objective functions. Such algorithms transform a multi-objective optimization problem into a sequence of single-objective optimization problems that have to be solved. Note that any MIL-MMP can be viewed as special cases of the problem of optimization a (non-linear) function over the set of efficient solutions of multi-objective mixed integer linear program. So, we call our algorithm a criterion space search algorithm because it attempts to solve a MIL-MMPs by working on the space of objective and solving a set of single-objective mixed integer linear programs using CPLEX. We conduct an extensive computational study and show the following results:

- The proposed algorithm outperforms CPLEX SOCP solver (for instances with $p \geq 2$) and the branch-and-bound algorithm proposed by Saghand et al. (2019) (for instances with $p = 2$) by a factor of more than 10 on many instances.

- Even if we linearize the objective function and solve the linearized problem by CPLEX, the proposed algorithm still performs significantly better, by a factor of more than 30 on many instances. In order to show this, we will conduct numerical experiments on instances of MIL-MMPs with only binary variables and $p = 2$. Evidently, linearization can be done easily for such instances compared to other variants of MIL-MMPs.

The remainder of the paper is organized as follows. In Section 2, we review some problems in different fields of study that can be formulated as MIL-MMPs. In Section 3, we provide some preliminaries for the proposed algorithm. In Section 4, a high-level description of the algorithm is provided. In Section 5, a detailed description of the proposed algorithm in given. In Section 6, we conduct an extensive computational study. Finally, in Section 7, some concluding remarks are explained.

2. Application

In this section, we present some problems in different fields of study that give rise to MIL-MMPs.

2.1. Game theory

In a recent study, Charkhgard et al. (2018) provided a detailed description of three applications of MIL-MMPs in game theory based on studies of Chakrabarty et al. (2006), Eisenberg and Gale

(1959), Jain and Vazirani (2007), and Vazirani (2012). These three applications are computing the Nash solution for a symmetric/non-symmetric bargaining problem, computing a market equilibrium in a *linear fisher market*, and computing market equilibrium to a Kelly capacity allocation market. In this section, we briefly review the Nash solution to a symmetric bargaining problem but interested researchers can refer to Charkhgard et al. (2018) for further details about all three applications.

A bargaining problem is a cooperative game in which all players agree to create a grand coalition, instead of competing with each other, to get a higher payoff (Serrano 2005). To be able to create a grand coalition, the agreement of all players is necessary. Therefore, a critical question to be answered is: What should the payoff of each player be in a grand coalition? One of the solutions to the (symmetric) bargaining problem was proposed by Nash and is now known as the Nash bargaining solution (Nash 1950, 1953).

We denote the expected utility of the players by $\boldsymbol{y} = (y_1, \cdots, y_p)$ where p is the number of players. Let $\mathcal{Y}$ be the p-dimensional feasible set in the payoff space and $\boldsymbol{q} = (q_1, \cdots, q_p)$ be the disagreement point, i.e., the payoffs that players will receive under no coalition. Nash proved that, under certain conditions, the optimal solution $\boldsymbol{y}^* = (y_1^*, \cdots, y_p^*)$ to the bargaining problem is the unique solution satisfying,

$$\boldsymbol{y}^* \in \arg\max \left\{ \prod_{i=1}^{p} (y_i - q_i) : \boldsymbol{y} \in \mathcal{Y},\ y_i \geq q_i\ \forall i \in \{1, \cdots, p\} \right\}.$$

2.2. Conservation planning

Another problem that can be formulated as a MIL-MMP is multiple-species conservation planning problem. A typical conservation planning problem is deciding which sites to protect within a geographical region to preserve biodiversity (Nicholson and Possingham 2006a). The feasible set of such a problem can be easily formulated using binary decision variables and some constraints (Haider et al. 2018). For example, one can use a binary decision variable for each site to indicate whether that site should be selected for protection or not. Also, budget constraints, connectivity constraints (i.e., the selected sites should be connected), and compactness constraints (i.e., the selected sites should form a compact shape) can be considered (Haider et al. 2018). However, the main issue is how to mathematically define the objective function, which is preserving the biodiversity. Many researchers argue that such a goal can be obtained by using extinction risks (see for instance Calkin et al. (2002), Nicholson and Possingham (2006b), and Williams and Araújo (2002)). Specifically, multiple-species conservation planning problem can be formulated as follows:

$$\max_{\boldsymbol{x}\in\mathcal{X}} \prod_{i=1}^{p} \big(1 - y_i(\boldsymbol{x})\big),$$

where $y_i(\boldsymbol{x})$ is the probability of extinction of species $i \in \{1,\dots,p\}$ for a feasible solution $\boldsymbol{x} \in \mathcal{X}$.

2.3. System reliability

Reliability is defined as the probability of a system/component performing a required function under stated operating conditions without failure for a given period of time. Maximizing the reliability for series-parallel systems have been studied by many researchers (see for instance Coit (2001), Ardakan and Hamadani (2014), and Feizabadi and Jahromi (2017)). Such systems include p subsystems connected in series. Each subsystem includes a number of active and standby components. Since each component has a predetermined cost and weight, and there often exist some budget constraints associated with the total cost and weight, decisions have to be made about which components should be selected. Overall, maximizing the reliability for series-parallel systems can be formulated as follows:

$$\max_{\boldsymbol{x}\mathcal{X}} \prod_{i=1}^{p} y_i(\boldsymbol{x}),$$

where $y_i(\boldsymbol{x})$ is the reliability of subsystem $i \in \{1,\dots,p\}$ for a feasible solution $\boldsymbol{x} \in \mathcal{X}$.

3. Preliminaries

A MIL-MMP can be stated as follows,

$$\begin{aligned} \max\ & \prod_{i=1}^{p} y_i \\ \text{s.t. } & \boldsymbol{y} = D\boldsymbol{x} + \boldsymbol{d} \\ & A\boldsymbol{x} \le \boldsymbol{b} \\ & \boldsymbol{x}, \boldsymbol{y} \ge \boldsymbol{0}, \quad \boldsymbol{x} \in \mathbb{R}^{n_c} \times \mathbb{B}^{n_b} \times \mathbb{Z}^{n_i}, \quad \boldsymbol{y} \in \mathbb{R}^{p}, \end{aligned} \tag{2}$$

where n_c, n_b, and n_i represent the number of continuous, binary, and integer decision variables, respectively. Also, D is a $p \times n$ matrix where $n := n_c + n_b + n_i$, $\boldsymbol{d}$ is a p-vector, A is an $m \times n$ matrix, and $\boldsymbol{b}$ is an m-vector. For notational convenience, we partition the index set of variables $\mathcal{N} := \{1, 2, ..., n\}$ into continuous $\mathcal{C}$, binary $\mathcal{B}$, and integer $\mathcal{I}$.

In this paper, we sometimes refer to the sets $\mathcal{X} := \{\boldsymbol{x} \in \mathbb{R}^{n_c} \times \mathbb{B}^{n_b} \times \mathbb{Z}^{n_i} : A\boldsymbol{x} \leq \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0}\}$ and $\mathcal{Y} := \{\boldsymbol{y} \in \mathbb{R}^p : \boldsymbol{x} \in \mathcal{X},\ \boldsymbol{y} = D\boldsymbol{x} + \boldsymbol{d},\ \boldsymbol{y} \geq \boldsymbol{0}\}$ as *the feasible set in the decision space* and *the feasible set in the criterion space*, respectively. 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., there exists a $\boldsymbol{y} \in \mathcal{Y}$ such that $\boldsymbol{y} > \boldsymbol{0}$. We usually 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).

As mentioned in Introduction, a MIL-MMP can be reformulated as a mixed integer SOCP. This procedure is explained in Charkhgard et al. (2018) based on the study of Ben-Tal and Nemirovski (2001). So, in this section, we first review how this reformulation can be done. Observe that by introducing a new non-negative variable, γ, and a *geometric mean* constraint, Problem (2) can be reformulated as follows:

$$\begin{aligned} &\max\ \gamma \\ &\text{s.t. } 0 \leq \gamma \leq \Big(\prod_{i=1}^{p} y_i\Big)^{\frac{1}{p}} \\ &\qquad \boldsymbol{y} \in \mathcal{Y}. \end{aligned}$$

It is evident that if $\bar{\gamma}$ is the optimal objective value of the reformulated problem, then $\bar{\gamma}^p$ is the optimal objective value of Problem (2). Let k be the smallest integer value such that $2^k \geq p$. By introducing a set of non-negative variables and constraints, the geometric mean constraint can be replaced as follows:

$$\begin{aligned} &\max\ \gamma \\ &\text{s.t. } 0 \leq \gamma \leq \Gamma \\ &\qquad 0 \leq \Gamma \leq \sqrt{\tau_1^{k-1}\tau_2^{k-1}} \end{aligned}$$

$$0 \leq \tau_j^l \leq \sqrt{\tau_{2j-1}^{l-1}\tau_{2j}^{l-1}} \qquad \text{for } j = 1,\dots,2^{k-l} \text{ and } l = 1,\dots,k-1$$

$$0 \leq \tau_j^0 = y_j \qquad \text{for } j = 1,\dots,p$$

$$0 \leq \tau_j^0 = \Gamma \qquad \text{for } j = p+1,\dots,2^k$$

$$\boldsymbol{y} \in \mathcal{Y}.$$

The above formulation is a mixed integer SOCP since any constraint of the form $\{u,v,w \geq 0 : u \leq \sqrt{vw}\}$ is equivalent to $\{u,v,w \geq 0 : \sqrt{u^2 + (\frac{v-w}{2})^2} \leq \frac{v+w}{2}\}$. Now, one can use a commercial mixed integer second-order cone programming solver such as CPLEX to solve any MIL-MMP. However, the goal of this paper is to develop an effective and different solution approach based on multi-objective optimization. Hence, the following definition and proposition are critical.

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*

$$y_i \leq y_i' \qquad \forall i \in \{1,2,...,p\}$$

$$y_i < y_i' \qquad \textit{for at least one } i \in \{1,2,...,p\}$$

where $\boldsymbol{y} := D\boldsymbol{x} + \boldsymbol{d}$ and $\boldsymbol{y}' := D\boldsymbol{x}' + \boldsymbol{d}$. 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* (2)*, denoted by $\boldsymbol{x}^*$, is an efficient solution and therefore its corresponding image in the criterion space, denoted by $\boldsymbol{y}^*$ where $\boldsymbol{y}^* := D\boldsymbol{x}^* + \boldsymbol{d}$, is a nondominated point.*

Proof. Suppose that $\boldsymbol{x}^*$ is an optimal solution of Problem (2) but it is not an efficient solution. By definition, this implies that there must exist a feasible solution denoted by $\boldsymbol{x} \in \mathcal{X}$ that dominates $\boldsymbol{x}^*$. In other words, we must have that

$$y_i^* \leq y_i \qquad \forall i \in \{1,2,...,p\}$$

$$y_i^* < y_i \qquad \text{for at least one } i \in \{1, 2, ..., p\}$$

where $\boldsymbol{y} := D\boldsymbol{x} + \boldsymbol{d}$. Also, by assumptions of Problem (2), we know that $\boldsymbol{y}^* > \boldsymbol{0}$. Therefore, we must have that $0 < \prod_{i=1}^{p} y_i^* < \prod_{i=1}^{p} y_i$. Consequently, $\boldsymbol{x}^*$ cannot be an optimal solution (a contradiction). Q.E.D.

Proposition 1 implies that Problem (2) is equivalent to $\max_{\boldsymbol{y}\in\mathcal{Y}_N} \prod_{i=1}^{p} y_i$. Therefore, this problem is precisely optimization over the efficient set. Hence, instead of searching the entire feasible set, we propose an algorithm which looks for an optimal solution by searching over the set of nondominated points.

As we will explain in Section 5.1, the proposed algorithm sometimes adds the so-called *no-good* constraints (Hooker 2011). It is known that no-good constraints for problems with general integer decision variables are naturally non-linear (Fischetti et al. 2005). Although it is possible to linearize such constraints, it is more convenient to add no-good constraints to problems involving only binary decision variables because they will be naturally linear. So, in the rest of this study, whenever we want to solve a MIL-MMP by the proposed algorithm, we assume that all integer variables are transformed into binary decision variables using the standard binary transformation procedure. Specifically, because we have assumed that $\mathcal{X}$ is bounded then for any integer decision variable x a global upper bound should be computable, i.e., $x \leq u$. Hence, the variable x can be replaced by introducing $v := \lfloor log_2 u \rfloor + 1$ new binary variables as follows,

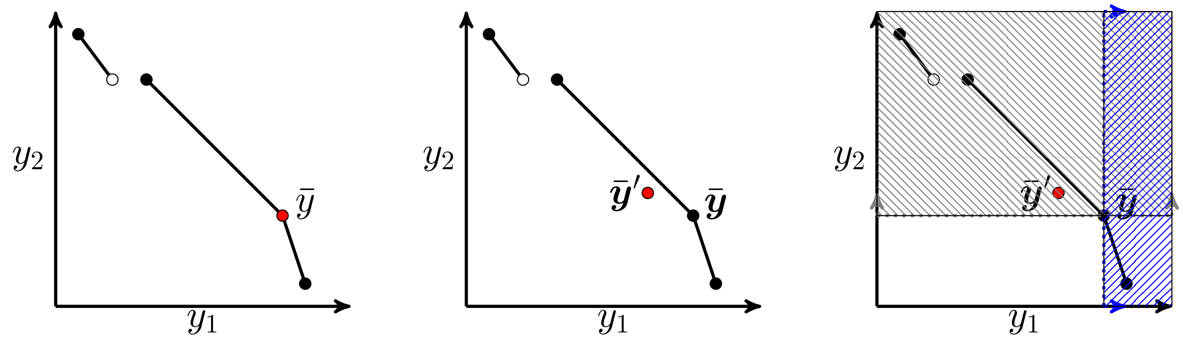
$$x := 2^0 z_1 + 2^1 z_2 + ... + 2^{v-1} z_v.$$

Also, we assume that the following constraint is also added when using the transformation,

$$2^0 z_1 + 2^1 z_2 + ... + 2^{b-1} z_b \leq u.$$

4. High-level description

In this section, we provide a high-level description of our algorithm. In each iteration, the algorithm starts by computing a *locally* nondominated point, denoted by $\bar{\boldsymbol{y}}$. Note that the term "locally" is used because in each iteration the algorithm removes a set of feasible solutions from the problem by adding some constraints. So, the point $\bar{\boldsymbol{y}}$ will be certainly a nondominated point for the restricted feasible set but not necessarily for the entire feasible set. An illustration of the nondominated frontier of Problem (2) when $p = 2$ and also $\bar{\boldsymbol{y}}$ (obtained in the first iteration) can be found in Figure 1a. As an aside, note that the nondominated frontier of a multi-objective mixed integer linear program can be a very complicated shape. For example, when $p = 2$, it may contain some continuous segments, points, open or half-open line segments (Boland et al. 2015).



(a) The nondominated frontier (b) The lower bound point (c) The decomposition

Figure 1 A high-level description of the proposed algorithm when $p = 2$

It is obvious that $\prod_{i=1}^{p} \bar{y}_i$ is a global lower bound for Problem (2). However, a better lower bound can be (possibly) computable based on a solution corresponding to $\bar{\boldsymbol{y}}$, denoted by $\bar{\boldsymbol{x}}$. In order to find a better lower bound, we set the value of the binary variables in Problem (2) to those in solution $\bar{\boldsymbol{x}}$. Obviously, the resulted problem is a continuous MIL-MMP. So, one can optimize such a problem to obtain a (possibly) better feasible point in the criterion space, denoted by $\bar{\boldsymbol{y}}'$. It is evident that $\prod_{i=1}^{p} \bar{y}_i \leq \prod_{i=1}^{p} \bar{y}'_i$. An illustration of $\bar{\boldsymbol{y}}'$ can be found in Figure 1b when $p = 2$.

Note that $\bar{\boldsymbol{y}}'$ is not necessarily a nondominated point. However, by construction, it will provide the best possible lower bound when binary decision variables are fixed to their corresponding values in $\bar{\boldsymbol{x}}$. So, the key idea of the algorithm is to add the so-called no-good constraint to the formulation for ensuring that the *binary support vector*, i.e., the vector of the values of binary decision variables, associated with $\bar{\boldsymbol{x}}$ will be excluded from the search (in the future iterations).

Another key component of the algorithm is that it will decompose the criterion space in each iteration based on $\bar{\boldsymbol{y}}$. Since $\bar{\boldsymbol{y}}$ is a nondominated point for the the restricted problem, one can immediately remove the dominated part of the criterion space associated with $\bar{\boldsymbol{y}}$ and continue the search in the remaining areas. The new areas will be p new boxes, an illustration of which are shown in Figure 1c when $p=2$, i.e., the box with $y_1 \geq \bar{y}_1$ and the box with $y_2 \geq \bar{y}_2$.

It is evident that one can just simply repeat the process explained above in new boxes to be able to find an optimal solution of Problem (2). However, in order to speed up the searching process, we employ a procedure to calculate a global upper/dual bound for the problem. In other words, in each iteration, when exploring a box, we first compute a dual bound. If the dual bound is not strictly better than the best lower/primal bound found then there is no need to search the box anymore. In order to obtain a dual bound, we can simply drop the integrality conditions in Problem (2).

5. Detailed description

In this section, we explain a detailed description of the proposed algorithm.

5.1. No-good constraints

As mentioned in Section 4, no-good constraints, also known as *tabu* constraints (Hooker 2011, Fischetti and Lodi 2003, Hamming 1950), play an important role in the proposed algorithm. In the proposed algorithm, we maintain a list of the so-called tabu feasible solutions, denoted by TABU. Note that we assume that all general integer decision variables of Problem (2) are transformed to binary decision variables. Therefore, solution $\boldsymbol{x}' \in$ TABU consists of two support vectors: continuous and binary. For each $\boldsymbol{x}' \in$ TABU, the algorithm will add the following no-good constraint to ensure that the binary support vector associated with $\boldsymbol{x}'$ will be excluded from the search,

$$\sum_{j \in \mathcal{S}^0(\boldsymbol{x}')} x_j + \sum_{j \in \mathcal{S}^1(\boldsymbol{x}')} (1 - x_j) \geq 1,$$

where $\mathcal{S}^0 := \{j \in \mathcal{B} : x'_j = 0\}$ and $\mathcal{S}^1 := \{j \in \mathcal{B} : x'_j = 1\}$.

5.2. Computing primal bound

By Proposition 1, any efficient solution is expected to be a high-quality feasible solution for Problem (2). So, it is expected that such a solution provides a good (global) lower/primal bound for the problem. Therefore, the algorithm attempts to compute a *locally* efficient solution in each iteration. In multi-objective optimization, it is well-known (see for instance Ehrgott (2005)) that optimizing any positive wighted summation of the objective functions over the feasible set must return an efficient solution (if exists any). So, in this research, we simply assume that the weights are equal to one, and therefore we optimize/maximize the corresponding objective function, i.e., $\sum_{i=1}^{p} y_i$, over the feasible set, i.e., $y \in \mathcal{Y}$, but with some additional constraints.

The additional constraints include some no-good constraints as well as imposing a lower bound for each objective function. Note that additional constraints (in particular the no-good constraints) do not guarantee that the solution found by solving the weighted sum optimization problem returns a true efficient solution for the corresponding multi-objective problem (with no additional constraints). However, an optimal solution (if exists any) must be obviously efficient for the *restricted* problem, i.e., the multi-objective optimization problem with the additional constraints. That is the reason that we call such an optimal solution a locally efficient solution.

In light of the above, the proposed algorithm solves the following (weighted sum) optimization problem, denoted by $\text{WSO}(\boldsymbol{l}, \textsc{Tabu})$, in each iteration to find a locally efficient solution (meaning a good lower/primal bound):

$$(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \in \arg\max \left\{ \sum_{i=1}^{p} y_i : \boldsymbol{y} \in \mathcal{Y},\ \boldsymbol{y} \geq \boldsymbol{l}, \quad \sum_{j \in \mathcal{S}^0(\boldsymbol{x}')} x_j + \sum_{j \in \mathcal{S}^1(\boldsymbol{x}')} (1 - x_j) \geq 1 \quad \forall \boldsymbol{x}' \in \textsc{Tabu} \right\},$$

where $\textsc{Tabu}$ is the list of tabu solutions that their corresponding binary support vectors should be excluded from the search. Also, $\boldsymbol{l} \in \mathbb{R}^p_+$ is the vector of lower bound values for objective functions $y_1, \ldots, y_p$. Note that as explained in Section 4, in each iteration, the algorithm needs to search a box defined by $\{\boldsymbol{y} \in \mathbb{R}^p : \boldsymbol{l} \leq \boldsymbol{y} \leq +\infty\}$. So, the vector $\boldsymbol{l}$ is basically restricting the search to a box in the above optimization problem.

After calling WSO($\boldsymbol{l}$,TABU) and computing $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$, if it is feasible then the algorithm will fix the value of integer decision variables by setting them equal to values of the binary support vector of $\bar{\boldsymbol{x}}$ in Problem (2),

$$(\bar{\boldsymbol{x}}', \bar{\boldsymbol{y}}') \in \arg\max \left\{ \prod_{i=1}^{p} y_i : \boldsymbol{y} \in \mathcal{Y},\ x_i = \bar{x}_i \quad \forall i \in \mathcal{B} \right\}.$$

Note that in this continuous optimization problem there exist no additional constraints (other than fixing constraints). This is important because it guarantees that by solving this continuous optimization problem we can compute the best possible solution for a fixed binary support vector. In order to solve this continuous optimization problem, we first transform it to a continuous SOCP (using the same procedure explained in Section 3) and then solve it by a suitable solver, e.g., CPLEX. We denote the operation of computing $(\bar{\boldsymbol{x}}', \bar{\boldsymbol{y}}')$ by SOCP-LB($\bar{\boldsymbol{x}}$).

5.3. Computing dual bound

We first make an important observation that can result in a (good) cut when computing a dual bound in each iteration. Employing such a cut is important because it can improve the dual bound value in practice.

Observation 1 *Let $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ be an optimal solution of* WSO($\boldsymbol{l}$,TABU) *at a particular iteration. Due to optimality, the following cut can be used when searching the box associated with $\boldsymbol{l}$ in the subsequent iterations,*

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

We now explain how to compute a dual bound. In each iteration, before finding a primal bound the algorithm first attempts to computer a dual bound (for its associated box $\boldsymbol{l}$) by solving the following continuous optimization problem,

$$(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) \in \arg\max \left\{ \prod_{i=1}^{p} y_i : \boldsymbol{y} \in \mathcal{Y}_R,\ \boldsymbol{y} \geq \boldsymbol{l},\ \sum_{j \in \mathcal{S}^0(\boldsymbol{x}')} x_j + \sum_{j \in \mathcal{S}^1(\boldsymbol{x}')} (1 - x_j) \geq 1 \quad \forall \boldsymbol{x}' \in \text{TABU}, \right.$$
$$\left. \sum_{i=1}^{p} y_i \leq \sum_{i=1}^{p} \bar{y}_i^{Parent} \right\},$$

where $\mathcal{Y}_R$ is basically a relaxation of $\mathcal{Y}$ that can be obtained by dropping the integrality condition on binary decision variables, i.e., $x_i \in [0,1]$ for $i \in \mathcal{B}$. Note that, as explained in Section 4, the current box (in each iteration) is obtained as a result of decomposing a larger box, referred to as *Parent* box, in one of the previous iterations. By this explanation, $\bar{y}_i^{Parent}$ is basically the feasible point that is obtained as the result of solving the weighted sum optimization problem for the parent box. So, $\sum_{i=1}^{p} y_i \leq \sum_{i=1}^{p} \bar{y}_i^{Parent}$ is the cut derived based on Observation 1.

In order to solve this continuous optimization problem, we again first transform it to a continuous SOCP solver (using the same procedure explained in Section 3) and then solve it by a suitable solver, e.g., CPLEX. We denote the operation of computing $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$ by SOCP-UB$(\boldsymbol{l}, \bar{\boldsymbol{y}}^{Parent}, \textsc{Tabu})$.

Algorithm 1: The proposed algorithm

```
1  Input: A feasible instance of Problem (2)
2  $Queue.create(\textsc{Tree})$
3  $List.create(\textsc{Tabu})$
4  $\boldsymbol{l} \leftarrow -\infty$; $\bar{\boldsymbol{y}}^{Parent} \leftarrow +\infty$
5  $\text{G}_{\text{LB}} \leftarrow -\infty$; $\text{G}_{\text{UB}} \leftarrow +\infty$
6  $\textsc{Tree}.add\big((\boldsymbol{l}, \bar{\boldsymbol{y}}^{Parent})\big)$
7  while $not\ Queue.empty(\textsc{Tree})$ & $\text{G}_{\text{UB}} - \text{G}_{\text{LB}} \geq \varepsilon_1$ & $\frac{\text{G}_{\text{UB}} - \text{G}_{\text{LB}}}{\text{G}_{\text{UB}}} \geq \varepsilon_2$ do
8      $\textsc{Tree}.PopOut\big((\boldsymbol{l}, \bar{\boldsymbol{y}}^{Parent})\big)$
9      $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) \leftarrow$ SOCP-UB$(\boldsymbol{l}, \bar{\boldsymbol{y}}^{Parent}, \textsc{Tabu})$
10     Update $\text{G}_{\text{UB}}$
11     if $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) \neq (null,null)$ & $\prod_{i=1}^{p} \tilde{y}_i > \text{G}_{\text{LB}}$ then
12         $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \leftarrow$ WSO$(\boldsymbol{l}, \textsc{Tabu})$
13         if $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \neq (null,null)$ then
14             $(\bar{\boldsymbol{x}}', \bar{\boldsymbol{y}}') \leftarrow$ SOCP-LB$(\bar{\boldsymbol{x}})$
15             if $\prod_{i=1}^{p} \bar{y}_i' > \text{G}_{\text{LB}}$ then
16                 $\text{G}_{\text{LB}} \leftarrow \prod_{i=1}^{p} \bar{y}_i'$
17                 $(\boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow (\bar{\boldsymbol{x}}', \bar{\boldsymbol{y}}')$
18             $\textsc{Tabu}.add(\bar{\boldsymbol{x}})$
19             foreach $i \in \{1, 2, .., p\}$ do
20                 $\boldsymbol{l}^i \leftarrow \boldsymbol{l}$
21                 $l_i^i \leftarrow \bar{y}_i$
22                 $\textsc{Tree}.add\big(\boldsymbol{l}^i, \bar{\boldsymbol{y}}\big)$
23 return $(\boldsymbol{x}^*, \boldsymbol{y}^*)$
```

5.4. The proposed algorithm

In Sections 5.1-5.3, we explained the key components of the algorithm. So, we can now present a precise description of the proposed algorithm which can also be found in Algorithm 1. The algorithm maintains a queue of nodes, denoted by Tree. Each node in the queue contains two vectors denoted by $\boldsymbol{l}$ and $\bar{\boldsymbol{y}}^{Parent}$. As mentioned before, $\boldsymbol{l}$ is a lower bound point in the criterion space that defines a box. Also, $\bar{\boldsymbol{y}}^{Parent}$ is the feasible point obtained by solving the weighted sum optimization problem for the parent of each node. We initialize the queue (of nodes) by $\boldsymbol{l} = -\boldsymbol{\infty}$ and $\bar{\boldsymbol{y}}^{Parent} = +\boldsymbol{\infty}$. The algorithm also maintains three other pieces of information including a tabu list which is denoted by $\textsc{Tabu}$, a global upper/dual bound denoted by G_{UB}, and a global lower/primal bound denoted by G_{LB}. At the beginning, $\textsc{Tabu}$ is empty and we set $\mathrm{G}_{\mathrm{UB}} = +\infty$ and $\mathrm{G}_{\mathrm{LB}} = -\infty$. The algorithm explores the queue as long as it is nonempty and $\mathrm{G}_{\mathrm{UB}} - \mathrm{G}_{\mathrm{LB}} \geq \varepsilon_1$ and $\frac{\mathrm{G}_{\mathrm{UB}} - \mathrm{G}_{\mathrm{LB}}}{\mathrm{G}_{\mathrm{UB}}} \geq \varepsilon_2$, where $\varepsilon_1, \varepsilon_2 \in (0,1)$ are the user-defined absolute and relative optimality gap tolerances, respectively. At the end, the algorithm returns the best solution found and its image in the criterion space, denoted by $(\boldsymbol{x}^*, \boldsymbol{y}^*)$. Next, we explain how each node of the queue is explored.

In each iteration, the algorithm pops out a node from the queue and denotes it by $(\bar{\boldsymbol{y}}, \boldsymbol{l})$. Note that when a node is popped out from the queue then that node does not exist in the queue anymore. Also, note that, in this study, we use the best-bound strategy to select a node for being popped out since we have numerically observed that this strategy performs the best. The algorithm next calls SOCP-UB$(\boldsymbol{l}, \bar{\boldsymbol{y}}^{Parent}, \textsc{Tabu})$ to compute a dual/upper bound of the node. The output of this operation is denoted by $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$. Next, the algorithm updates the G_{UB} accordingly. Afterwards, if $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) \neq (null, null)$, i.e., the relaxation is feasible, and $\prod_{i=1}^{p} \tilde{y}_i > \mathrm{G}_{\mathrm{LB}}$ then the algorithm attempts to find a primal bound and create new nodes (if possible) as follows.

The algorithm first calls WSO$(\boldsymbol{l}, \textsc{Tabu})$ to compute $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$. If $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) = (null, null)$, i.e., no feasible solution exists, then no further exploration is required and so a new iteration should be started (if possible). Otherwise, the algorithm calls SOCP-LB$(\bar{\boldsymbol{x}})$ to compute $(\bar{\boldsymbol{x}}', \bar{\boldsymbol{y}}')$, i.e., the best possible primal bound for the binary support vector associated with $\bar{\boldsymbol{x}}$. If the obtained solution is better than the best global solution then the algorithm updates G_{LB} and $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ accordingly. Next, $\bar{\boldsymbol{x}}$ will be added to the tabu list, and p new nodes will be created. The difference between the new nodes are simply their associated boxes. Specifically, for each $i \in \{1, \ldots, p\}$, the node $(\boldsymbol{l}^i, \bar{\boldsymbol{y}})$ will be added where $\boldsymbol{l}^i$ defines the associated box such that $l^i_j = l_j$ for each $j \in \{1, \ldots, p\} \backslash \{i\}$ and $l^i_i = \bar{y}_i$.

Observation 2 *The proposed algorithm is finite. This observation is true due to the following two reasons: (1) In each iteration a new binary support vector will be added to the tabu list by construction; and (2) The number of binary support vectors is finite because $\mathcal{X}$ is bounded by assumptions.*

6. Computational study

In this section, we compare the performance of our proposed algorithm (referred to as Algorithm 1) with the performance of the mixed integer SOCP solver of CPLEX (referred to as SOCP) and the algorithm proposed in our recent study (Saghand et al. 2019) for instances with $p = 2$ (referred to as B&B). We implement our algorithm in C++ and use CPLEX 12.7 through this computational study. The instances and the C++ implementation of our algorithm used in this computational study can be found in `https://goo.gl/otpJwX` and `https://goo.gl/KLWSB2`, respectively. The computational experiments are conducted on a Dell PowerEdge R360 with two Intel Xeon E5-2650 2.2 GHz 12-Core Processors (30MB), 128GB RAM, the RedHat Enterprise Linux 6.8 operating system, and using a single thread. Both the absolute and relative optimality gap tolerances, ε_1 and ε_2, are set to 10^{-6} in this computational study. Also, a time limit of 7200 seconds is imposed for solving each instance for all algorithms. Next, we explain how the instances are generated.

In our computational study, we test the performance of our algorithm on instances with $p \in \{2, 3, 4\}$ objective functions. For any $p \in \{2, 3, 4\}$, we generate four classes of instances including pure integer instances ($n_i = n$), pure binary instances ($n_b = n$), mixed integer instances ($n_i = 0.5n$ and $n_c = 0.5n$), and mixed binary instance ($n_b = 0.5n$ and $n_c = 0.5n$). Each class contains 16 subclasses of instances based on the dimensions of the matrix $A_{m\times n}$, and each subclass contains 10 instances. So, each class has a total of 160 instances and the total number of instances used in this computational study is $3 \times 4 \times 160 = 1920$.

We assume that $m \in \{200, 400, 800, 1600\}$ and $n = \alpha m$ where $\alpha \in \{0.5, 1, 1.5, 2\}$. For example, the subclass 200×100 implies that $m = 200$ and $n = 100$, i.e., $\alpha = 0.5$. The sparsity of matrix A is set to 50%. The components of vector $\boldsymbol{b}$ and the entries of matrix A are randomly drawn from discrete uniform distributions $[50, 150]$ and $[10, 30]$, respectively. We set the components of vector $\boldsymbol{d}$ equal to zero. The sparsity of each row of the matrix D was also set to 50%, and its components were drawn randomly from a discrete uniform distribution $[1, 10]$. Note that, since all constraints of the set $\mathcal{X}$ are inequality constraints and all coefficients of matrix A are nonnegative, the set $\mathcal{X}$ is bounded.

As mentioned in Section 3, each integer decision variable should be converted to (a set of) binary decision variables when using our proposed algorithm, i.e., Algorithm 1. Note that when solving an instance with other methods, no transformation is required. Since the maximum value for components of vector $\boldsymbol{b}$ is 150 and the minimum value for entries of matrix A is 10, each decision variable can get a maximum value of 15. Therefore, in the process of converting integer variables to binary variables, the integer variable x_i is converted to four binary variables as follows,

$$x_i := z_{i_1} + 2z_{i_2} + 4z_{i_3} + 8z_{i_4},$$

$$z_{i_1} + 2z_{i_2} + 4z_{i_3} + 8z_{i_4} \leq 15.$$

Note that the additional constraint is redundant, and there is no need to be added. Hence, for a pure integer instance, the number of decision variables increases by a factor of 4 when using Algorithm 1.

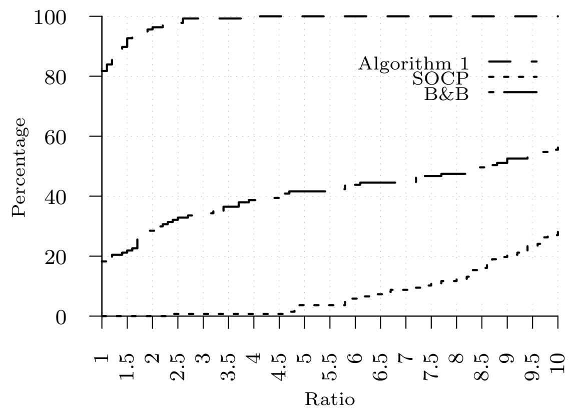
It is worth mentioning that in this computational study, we frequently use *performance profiles* (Dolan and Moré 2002) to compare different algorithms in terms of their solution times. A performance profile presents cumulative distribution functions for a set of algorithms being compared with respect to a specific performance metric. 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, 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.

6.1. Two objectives ($p = 2$)

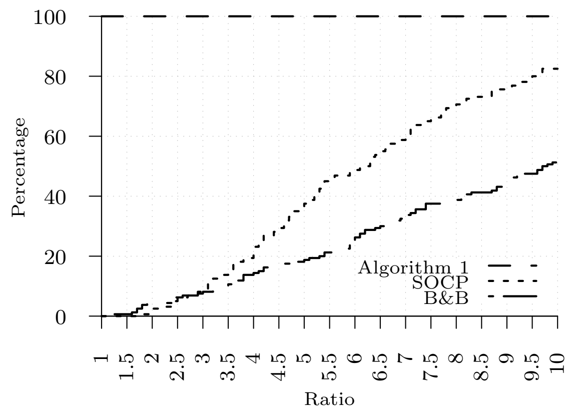
In this section, we compare the performance of Algorithm 1, SOCP, and B&B. Note that because B&B is developed for instances with $p = 2$, we only use B&B on this section. Figure 2 shows the run time performance profiles of all three approaches for different classes of instances when $p = 2$. From Figure 2, it is evident that the proposed algorithm significantly outperforms other algorithms. For example, for pure binary instances, Algorithm 1 outperforms SOCP and B&B by a factor of at least 10 on more than 45% and 75% of instances, respectively. Observe that, for instances involving general integer decision variables, the performance of Algorithm 1 decreases because of transforming integer variables to binary variables which results in increasing the size of an instance. However, Algorithm 1 is still significantly better than other approaches. For example, for mixed integer instances, Algorithm 1 outperforms B&B and SOCP by a factor of at least 2.5 on more than 40% and 60% of instances, respectively.

Tables 1-2 provide a detailed comparison between SOCP and Algorithm 1. Interested readers may refer to Appendix A for the detailed comparison between the B&B and Algorithm 1. In these tables, '#N' is the number of nodes explored, 'T(sec.)' shows the solution time in seconds, '#S' is the number of instances for which at least a feasible solution (other than zero) is found, and finally, '%G' shows the (relative) optimality gap percentage. Note that, in the tables, the numbers are the averages over 10 instances. In cases that an algorithm was not able to find a feasible solution for all 10 instances (in a subclass), the numbers are the averages over #S. If column #S is not reported for a method then the method was able to find at least one feasible solution (other than zero) for all instances. Similarly, if column %G is not reported for a method then the method was able to solve all instances to optimality.

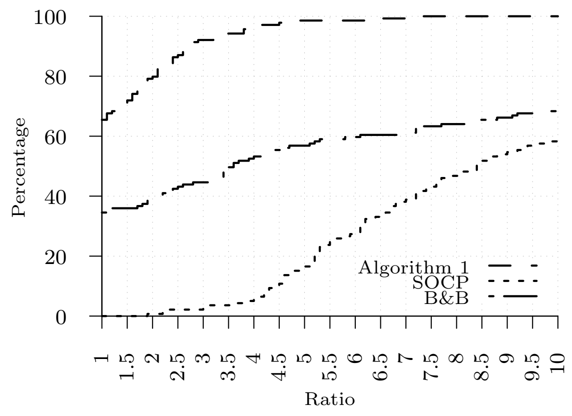
From Tables 1-2, we observe that, in pure binary and pure integer instances, there are some cases in which SOCP fails to find even a feasible solution (other than zero). According to IBM ILOG CPLEX Optimization Studio (2016), there is a parameter to change the convergence tolerance. Changing this tolerance to a smaller value may result in greater numerical precision of the solution,



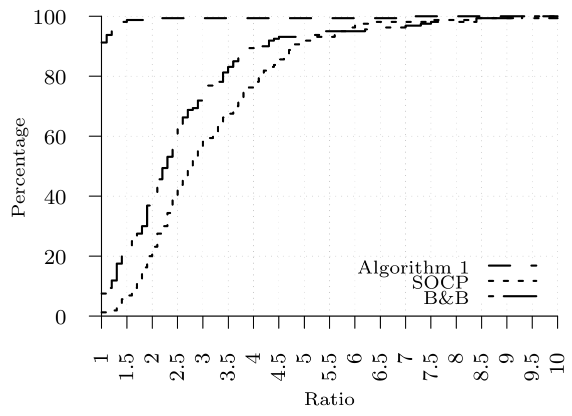
(a) Pure binary



(b) Mixed binary



(c) Pure integer



(d) Mixed integer

Figure 2 Performance profiles for instances with $p = 2$

but also increases the chance of a convergence failure in the algorithm and consequently may result in no solution at all. According to the manual, the smallest value for this tolerance is 10^{-12} and its proposed value is 10^{-7}. In order to overcome the convergence problem, we examined different values for this parameter, i.e., $\{10^{-6}, 10^{-7}, 10^{-8}, \cdots, 10^{-12}\}$, during the course of this study. However, we employed 10^{-6} and 10^{-12} in this paper because these values perform the best. Specifically, in this computational study, the default value is 10^{-12}. However, SOCP fails when solving pure binary and pure integer instances. Consequently, for those instances, we also provide results when the parameter is set to 10^{-6}. Note that, since SOCP fails (especially when the convergence parameter is 10^{-12}) to solve some of the instances, for any performance profile in this computational study, the graphs are drawn only based on the instances that are solved by all algorithms.

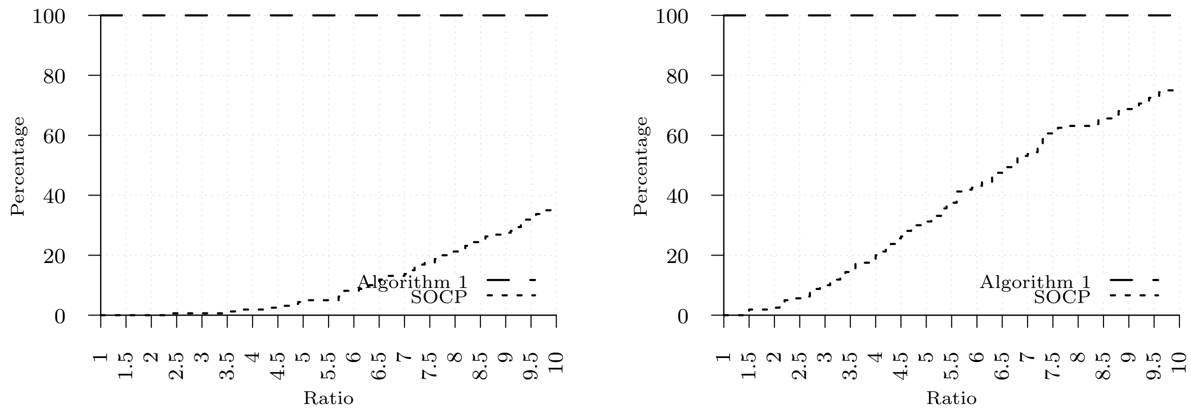
Table 1 Performance comparison between Algorithm 1 and SOCP on pure binary and mixed binary instances with $p = 2$

	Pure binary					Mixed binary			
	Algorithm 1		SOCP			Algorithm 1		SOCP	
$m \times n$	**#N**	**T(sec.)**	**#S**	**%G**	**T(sec.)**	**#N**	**T(sec.)**	**%G**	**T(sec.)**
200×100	6.80	1.02	9	0	7.96	5.20	0.67	0	1.98
200×200	6.40	2.16	10	0	23.26	4.40	1.04	0	5.11
200×300	6.20	4.90	10	0	60.88	5.20	1.74	0	8.37
200×400	8	10.88	10	0	100.76	5.20	2.32	0	11.31
Avg	6.85	4.74	9.75	0	49.24	5	1.44	0	6.69
400×200	6.60	5.58	8	0	60.63	4.40	3.03	0	14.79
400×400	7.20	21.76	8	0	273.84	4.40	5.53	0	29.46
400×600	6.40	35.83	10	0	568.04	4.80	8.75	0	47.47
400×800	6.20	73.57	9	0	1,124.32	4.40	10.88	0	76.51
Avg	6.60	34.19	8.75	0	527.85	4.50	7.05	0	42.06
600×300	8.20	22.37	7	0	343.48	4.60	6.82	0	35.01
600×600	8.20	116.95	8	0	1,845.93	4.80	14.21	0	107.03
600×900	6.60	187.37	6	0	3,560.81	4.60	22.41	0	178.79
600×1200	5.60	228.34	9	30	6,458.21	4.80	31.61	0	314.88
Avg	7.15	138.76	7.50	9	3,222.02	4.70	18.76	0	158.93
800×400	6.40	31.45	6	0	649.27	4.60	15.72	0	94.90
800×800	8.60	227.91	7	6	4,347.04	4.80	31.75	0	281.16
800×1200	9.80	752.30	10	53	7,200	4.40	48.53	0	504.64
800×1600	7.40	997.99	10	66	7,200	4.60	72.14	0	807.48
Avg	8.05	502.41	8.25	37.33	5,403.78	4.60	42.04	0	422.04

Table 2 Performance comparison between Algorithm 1 and SOCP on pure integer and mixed integer instances with $p = 2$

	Pure integer					Mixed integer			
	Algorithm 1		SOCP			Algorithm 1		SOCP	
$m \times n$	**#N**	**T(sec.)**	**#S**	**%G**	**T(sec.)**	**#N**	**T(sec.)**	**%G**	**T(sec.)**
200×100	7.20	2.45	9	0	8.61	5	1.09	0	2.61
200×200	7	7.42	9	0	32.63	5.20	2.51	0	5.79
200×300	6.40	10.29	10	0	57.54	4.60	3.49	0	9.10
200×400	6.80	17.36	10	0	99.85	4.60	4.77	0	12.79
Avg	6.85	9.38	9.50	0	51.19	4.85	2.97	0	7.57
400×200	5.80	11.83	10	0	67.18	4.60	5.51	0	14.46
400×400	7.20	51.10	10	0	270.70	4.80	12.59	0	26.43
400×600	6.80	80.70	8	0	498.25	4.60	20.07	0	38.32
400×800	7.40	145.43	10	0	1,183.91	4.20	25.87	0	64.15
Avg	6.80	72.27	9.50	0	505.36	4.55	16.01	0	35.84
600×300	8.40	50.96	7	0	315.68	4.20	13.02	0	30.04
600×600	5.80	145.57	9	0	1,531.51	4.60	34.46	0	116.49
600×900	6.80	283.66	7	5	3,595.14	4.80	63.91	0	177.41
600×1200	8	641.92	8	42	7,078.90	4.60	84.89	0	306.37
Avg	7.25	280.53	7.75	12.11	3,154.53	4.55	49.07	0	157.58
800×400	6.80	90.88	7	0	892.74	4.20	29.66	0	125.80
800×800	11.40	684.77	7	26	6,239.71	4.60	76.56	0	301.19
800×1200	10.20	1,071.83	10	51	7,180.49	4.40	133.84	0	513.50
800×1600	9.20	1,764.42	10	65	7,200	4.40	195.79	0	659.47
Avg	9.40	902.97	8.50	39.47	5,698	4.40	108.96	0	399.99

In light of the above, Figure 3 and Table 3 show the results on pure binary and pure integer instances when the parameter is set to 10^{-6}. From Figure 3, we observe that our proposed algorithm solves almost 65% of the pure binary instances and almost 25% of the pure integer instances 10 times faster than SOCP. Also, Table 3 shows that SOCP was able to find a feasible solution (other than zero) for all pure binary and pure integer instances when the parameter is set to 10^{-6}.



(a) Pure binary (b) Pure integer

Figure 3 Performance profile of instances with $p = 2$ for the convergence tolerance of 10^{-6}

Table 3 Performance comparison between Algorithm 1 and SOCP on instances with $p = 2$ for the convergence tolerance of 10^{-6}

	Pure binary					Pure integer				
	Algorithm 1		SOCP			Algorithm 1		SOCP		
$m \times n$	**#N**	**T(sec.)**	**#S**	**%G**	**T(sec.)**	**#N**	**T(sec.)**	**#S**	**%G**	**T(sec.)**
200×100	6.80	1.02	10	0	6.47	7.20	2.45	10	0	6.03
200×200	6.40	2.16	10	0	19.99	7	7.42	10	0	28.12
200×300	6.20	4.90	10	0	49.93	6.40	10.29	10	0	50.21
200×400	8	10.88	10	0	86.83	6.80	17.36	10	0	84.33
Avg	6.85	4.74	10	0	40.80	6.85	9.38	10	0	42.17
400×200	6.60	5.58	10	0	60.82	5.80	11.83	10	0	59.17
400×400	7.20	21.76	10	0	224.39	7.20	51.10	10	0	254.79
400×600	6.40	35.83	10	0	460.68	6.80	80.70	10	0	504.09
400×800	6.20	73.57	10	0	1,079.14	7.40	145.43	10	0	1,129.98
Avg	6.60	34.19	10	0	456.26	6.80	72.27	10	0	487.01
600×300	8.20	22.37	10	0	352.47	8.40	50.96	10	0	273.73
600×600	8.20	116.95	10	0	1,699.75	5.80	145.57	10	0	1,581.25
600×900	6.60	187.37	10	0	3,968.30	6.80	283.66	10	6	4,196.68
600×1200	5.60	228.34	10	28	6,446.24	8	641.92	10	36	7,006.16
Avg	7.15	138.76	10	7	3,116.69	7.25	280.53	10	10.5	3,264.45
800×400	6.40	31.45	10	0	638.13	6.80	90.88	10	0	746.86
800×800	8.60	227.91	10	0	3,957.99	11.40	684.77	10	10	5,505.12
800×1200	9.80	752.30	10	54	7,200	10.20	1,071.83	10	51	7,055.02
800×1600	7.40	997.99	10	65	7,200	9.20	1,764.42	10	64	7,200
Avg	8.05	502.41	10	29.75	4,749.03	9.40	902.97	10	31.25	5,126.75

6.2. Three objectives ($\mathbf{p = 3}$)

The run time performance profiles of Algorithm 1 and SOCP for pure binary and mixed binary instances with $p = 3$ are shown in Figures 4a and 4b, respectively. The table containing the detailed comparison can be found in Appendix B. We observe from Figure 4a that our proposed algorithm has solved almost 50% of the pure binary instances at least 10 times faster. Also, it was able to solve 20% of the mixed binary instances at least 5 times faster. We note that, among 160 instances with pure binary variables, SOCP was able to find a feasible solution (other than zero) for only 54 instances and only 40 (out of 54) instances were solved to optimality.

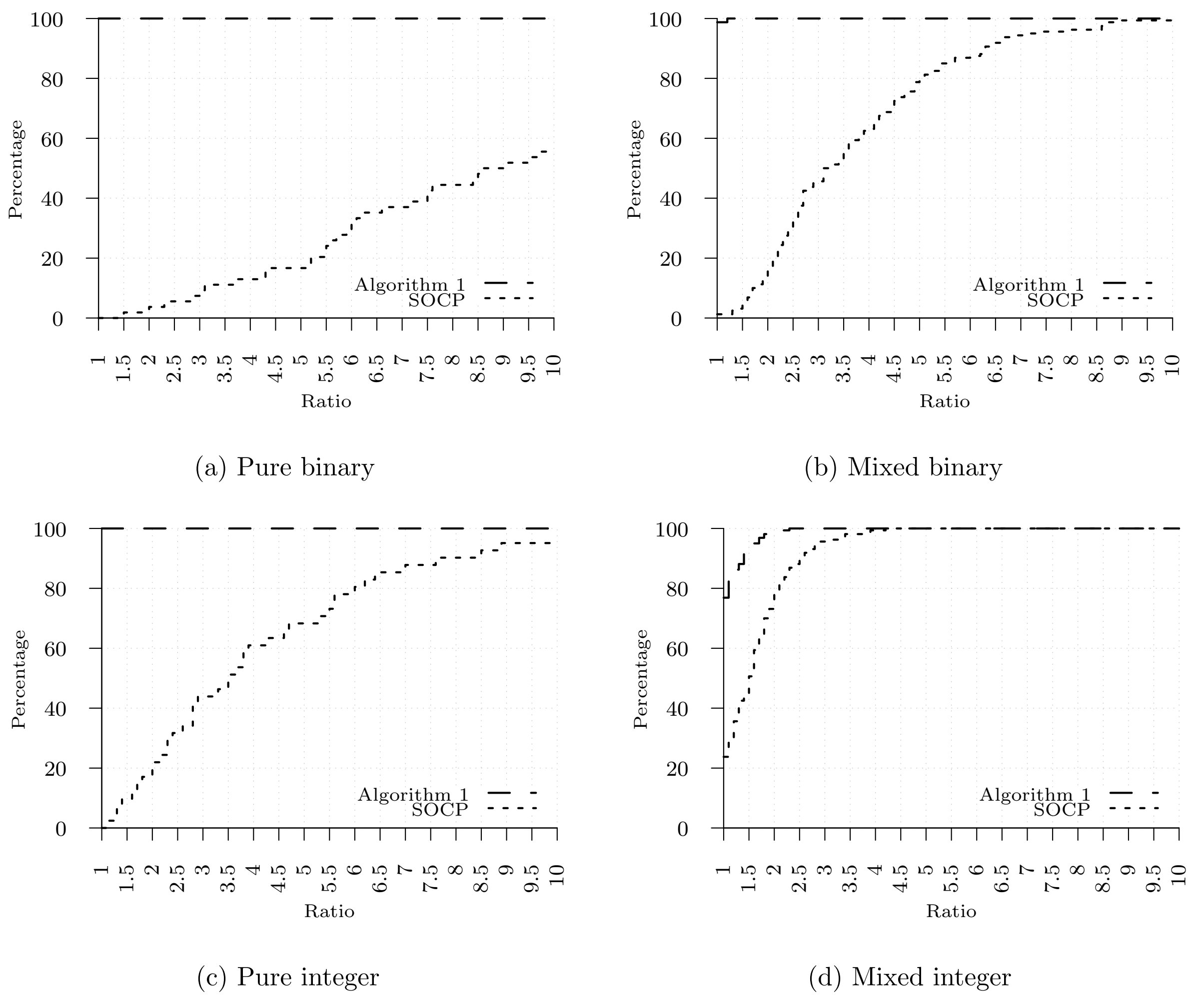
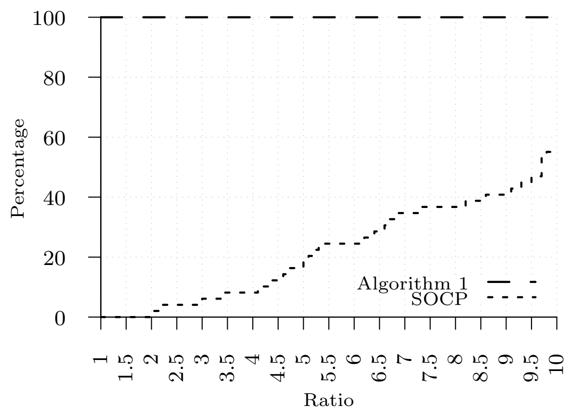


Figure 4 Performance profile of instances with $p = 3$

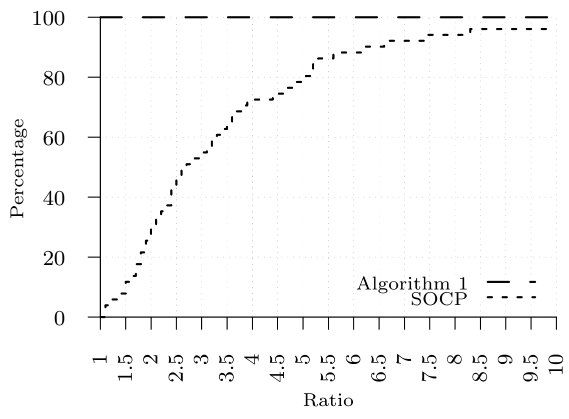
Figures 4c and 4d illustrate the run time performance profiles for instances with pure integer and mixed integer variables, respectively. The table containing the detailed comparison can be found in Appendix B. Observe that although our proposed algorithm requires to solve larger-sized instances (because of transformation), it has solved 20% of the pure integer instances 6 times faster.

Similarly, we observe that our proposed algorithm has solved 20% of the mixed integer instances 2 times faster. Also, it is worth mentioning that SOCP was able to find a feasible solution (other than zero) for only 41 out of 160 instances.

Figure 5 shows the run time performance profiles of the two algorithms for pure binary and pure integer instances when convergence tolerance is set to 10^{-6}. The table containing the detailed comparison can be found in Appendix B. Again, we observe that in both cases, our proposed algorithm outperforms SOCP. As shown in Figure 5a, Algorithm 1 has solved almost 45% of the pure binary instances 10 times faster than SOCP. Similarly, from Figure 5b, we observe that our algorithm solved 20% of the instances 5 times faster than SOCP.



(a) Pure binary

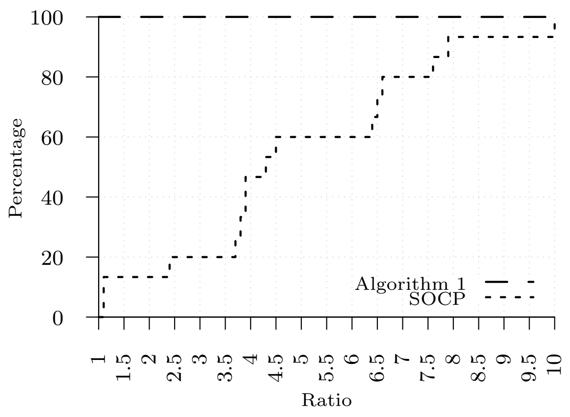


(b) Pure integer

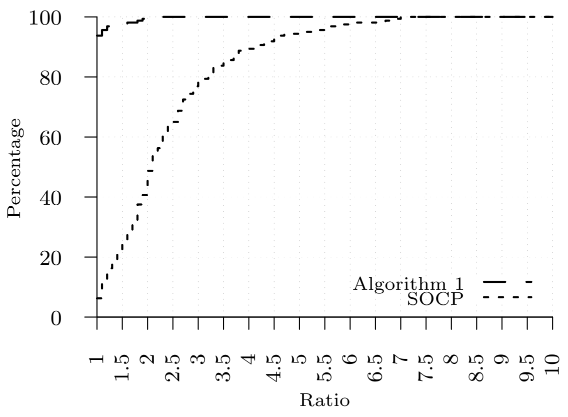
Figure 5 Performance profile of instances with $p = 3$ for the convergence tolerance of 10^{-6}

6.3. Four objectives $(\mathbf{p} = 4)$

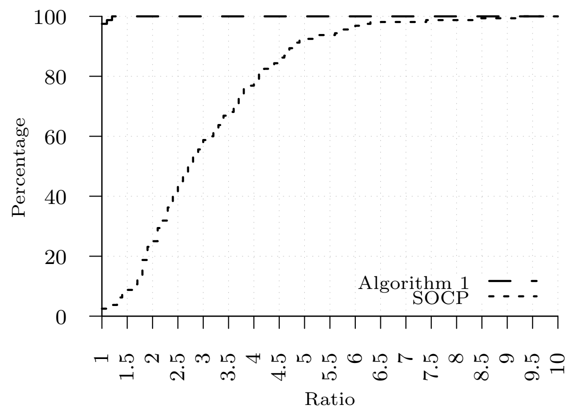
In this section, we study the performance of Algorithm 1 and SOCP for instances with $p = 4$. As an aside, we note that during the course of this study, we realized that as p increases, SOCP fails on more instances, i.e., it cannot find even a feasible solution (other than zero), when the convergence tolerance parameter is set to 10^{-12}. Specifically, for pure binary instances, SOCP was able to find a feasible solution (other than zero) for only 137, 54, and 15 instances (out of 160 instances) when $p = 2$, $p = 3$, and $p = 4$, respectively. For pure integer instances, SOCP was able to find a feasible solution (other than zero) for only 141, 41, and 12 instances (out of 160 instances) when $p = 2$, $p = 3$, and $p = 4$, respectively. However, Algorithm 1 has been always successful in finding a feasible solution (other than zero). In fact, Algorithm 1 was not able to solve only 4 out of 1920 instances (in this study) to optimality within the time limit. Those 4 instances are pure integer with $p = 4$ and even for them the optimality gap is less than 15%.



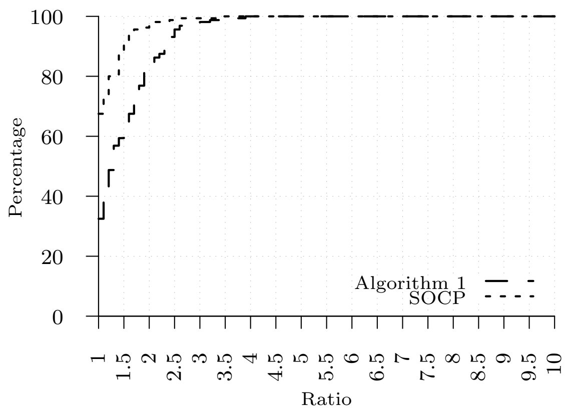
(a) Pure binary



(b) Mixed binary



(c) Pure integer



(d) Mixed integer

Figure 6 Performance profile of instances with $p = 4$

Figures 6a and 6b show the run time performance profiles of Algorithm 1 and SOCP on instances with pure binary and mixed binary variables, respectively. The table containing the detailed comparison can be found in Appendix C. As shown in Figure 6a, our proposed algorithm has solved 20% of the instances with pure binary variables at least 7.5 times faster. Similarly, Figure 6b shows that our algorithm has solved 20% of the instances with mixed binary instances at least 3 times faster.

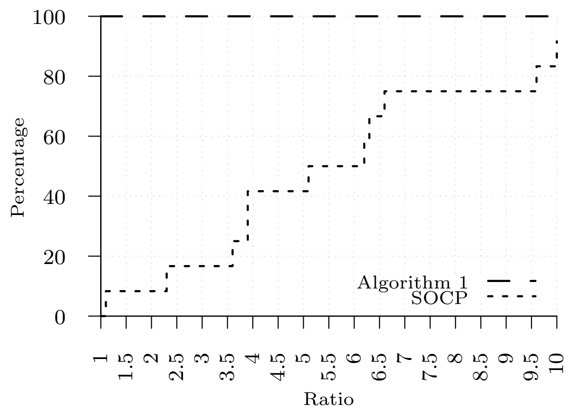
Figures 6c and 6d provide the run time performance profiles of the two solution methods on instances with pure integer and mixed integer variables, respectively. The table containing the detailed comparison can be found in Appendix C. From Figure 6c, we observe that our proposed algorithm solves almost 20% of the instances with pure integer variables at last 4 times faster. However, from Figure 6d, we observe that for the instances with mixed integer variables, SOCP outperforms Algorithm 1. It has solved almost 20% of the instances at least 2 times faster than our algorithm. However, there is an important point that should be considered for this observation. Based on the detailed comparison of the two algorithms, given in Appendix C (Table 11), we can observe that SOCP performs better for only small-sized instances, and as the size of the instances increases, our algorithm tends to solve instances faster. As an example, for the instances with 200 constraints, the average times are 9.43 seconds and 13.41 seconds for SOCP and Algorithm 1, respectively. However, for the instances with 800 constraints, the average times are 545.91 seconds and 478.69 seconds for SOCP and Algorithm 1, respectively.

Figures 7a and 7b show the run time performance profiles for pure binary and pure integer instances, respectively, when the convergence toleter is set to 10^{-6} for SOCP. The table containing the detailed comparison can be found in Appendix C. From Figure 7, we observe that our algorithm solves 20% of the pure binary instances and 20% of the pure integer instances at least 9.5 and 3.5 times faster, respectively.

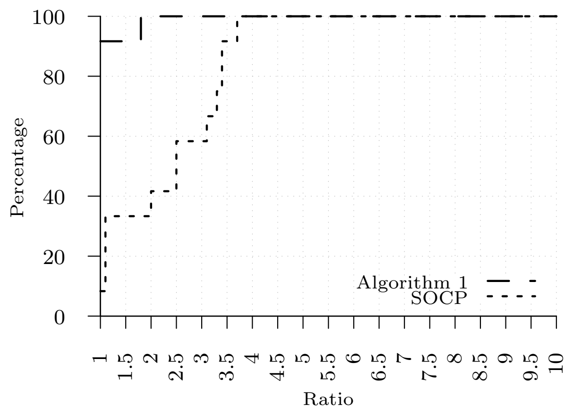
6.4. Linearization

Observe that pure binary and pure integer instances can be linearized and be solved by integer programming solvers such as CPLEX. It is evident that linearizing pure binary instances is less complicated. Also, linearizing instances with $p=2$ is easier. Consequently, the focus of this section will be only on pure binary instances with $p=2$. As an example, consider the following objective function,

$$\max\ (x_1+x_2)(x_2+x_3) = x_1x_2 + x_1x_3 + x_2^2 + x_2x_3$$



(a) Pure binary



(b) Pure integer

Figure 7 Performance profile of instances with $p = 4$ and the convergence tolerance of 10^{-6}

where x_1, x_2, and x_3 are binary variables. First, it is obvious that $x_i^2 = x_i$. Second, any bi-linear term $x_i x_j$ where both x_i and x_j are binary variables can be linearized by adding the following two constraints and introducing a new binary variable q,

$$2q \le x_i + x_j$$

$$x_i + x_j - 1 \le q$$

Note that, since the problem is in maximization form, the second constraint is redundant and can be eliminated. In light of this observation, the linearized objective function is as follows,

$$\begin{aligned}
\max\ & x_2 + q_1 + q_2 + q_3 & \\
\text{s.t. } & 2q_1 \le x_1 + x_2 & (x_1 x_2) \\
& 2q_2 \le x_1 + x_3 & (x_1 x_3) \\
& 2q_3 \le x_2 + x_3 & (x_2 x_3) \\
& q_i \in \{1, 0\} & \forall i \in \{1, 2, 3\}
\end{aligned}$$

We linearized all pure binary instances with $p = 2$ in the same way discussed above. Table 4 provides the detailed comparison between Algorithm 1, SOCP, and linearization approach for the pure binary instances with $p = 2$. Observe that the performance of linearization method is even worse than SOCP. In many instances, CPLEX was unable to find a feasible solution (other than 0) for linearized models. Note that, pure binary instances with $p = 2$ are in some sense the easiest class of instances for linearization. So, it is expected that the performance of the linearization technique to be even worse for other classes or larger values of p.

Table 4 Performance comparison between Algorithm 1, SOCP, and linearization method on pure binary instances with $p = 2$

	Pure binary							
	Algorithm 1		SOCP			Linearization		
$m \times n$	**#N**	**T(sec.)**	**#S**	**%G**	**T(sec.)**	**#S**	**%G**	**T(sec.)**
200×100	6.80	1.02	9	0	7.96	10	0	10.19
200×200	6.40	2.16	10	0	23.26	10	0	270.82
200×300	6.20	4.90	10	0	60.88	10	0	2,131.75
200×400	8	10.88	10	0	100.76	10	37	6,116.51
Avg	6.85	4.74	9.75	0	49.24	10	9.25	2,132.32
400×200	6.60	5.58	8	0	60.63	10	0	140.56
400×400	7.20	21.76	8	0	273.84	10	7	3,320
400×600	6.40	35.83	10	0	568.04	10	96	7,200
400×800	6.20	73.57	9	0	1,124.32	10	97	7,200
Avg	6.60	34.19	8.75	0	527.85	10	50	4,465.14
600×300	8.20	22.37	7	0	343.48	10	0	554.54
600×600	8.20	116.95	8	0	1,845.93	10	75	6,918.70
600×900	6.60	187.37	6	0	3,560.81	9	97	7,200
600×1200	5.60	228.34	9	30	6,458.21	0	-	-
Avg	7.15	138.76	7.50	9	3,222.02	7.25	55.96	4,811.46
800×400	6.40	31.45	6	0	649.27	10	0	1020.14
800×800	8.60	227.91	7	6	4,347.04	10	96	7,200
800×1200	9.80	752.30	10	53	7,200	1	98	7,200
800×1600	7.40	997.99	10	66	7,200	0	-	-
Avg	8.05	502.41	8.25	37.33	5,403.78	5.25	50.38	4,257.2

7. Final remarks

We developed a multi-objective mixed binary linear programming based algorithm for solving MIL-MMPs. This class of optimization problems has only one objective function and can be reformulated as mixed integer second-order cone programs. The reformulation can be directly solved by a commercial solver such as CPLEX. Using a detailed computational study, we demonstrated that our proposed algorithm significantly outperforms such a solver. We also showed that our algorithm outperforms a recent algorithm for solving instances of MIL-MMPs with $p = 2$. Finally, we showed that even by linearizing the objective function and solving the resulted (mixed) integer linear program by a commercial solver, the solution time will be significantly larger than the one obtained by our proposed approach.

Appendix A: Algorithm 1 vs B&B when $p = 2$

A detailed performance comparison between our algorithm and B&B (Saghand et al. 2019) on the instances with $p = 2$ can be found in Tables 5 and 6.

Table 5 Performance comparison between Algorithm 1 and B&B on pure binary and mixed binary instances with $p = 2$

	Pure binary					Mixed binary			
	Algorithm 1		B&B			Algorithm 1		B&B	
$m \times n$	**#N**	**T(sec.)**	**#S**	**%G**	**T(sec.)**	**#N**	**T(sec.)**	**%G**	**T(sec.)**
200×100	6.80	1.02	10	0	40.05	5.20	0.67	0	1.95
200×200	6.40	2.16	10	0	73.67	4.40	1.04	0	7.13
200×300	6.20	4.90	10	0	743.49	5.20	1.74	0	12.44
200×400	8	10.88	10	0	2,955.39	5.20	2.32	0	20.42
Avg	6.85	4.74	10	0	953.15	5	1.44	0	10.48
400×200	6.60	5.58	10	0	288.37	4.40	3.03	0	15.93
400×400	7.20	21.76	10	0	3,071.54	4.40	5.53	0	51.15
400×600	6.40	35.83	10	0	772.04	4.80	8.75	0	97.61
400×800	6.20	73.57	10	3	2,923.57	4.40	10.88	0	210.09
Avg	6.60	34.19	10	0.75	1,763.88	4.50	7.05	0	93.70
600×300	8.20	22.37	10	0	1,189.86	4.60	6.82	0	44.51
600×600	8.20	116.95	10	2	4,340.54	4.80	14.21	0	186.53
600×900	6.60	187.37	10	3	5,066.02	4.60	22.41	0	357.60
600×1200	5.60	228.34	10	2	5,079.97	4.80	31.61	0	621.89
Avg	7.15	138.76	10	1.75	3,919.10	4.70	18.76	0	302.63
800×400	6.40	31.45	10	0	229.88	4.60	15.72	0	114.58
800×800	8.60	227.91	10	15	6,487.76	4.80	31.75	0	457.34
800×1200	9.80	752.30	9	44	7,200	4.40	48.53	0	964.60
800×1600	7.40	997.99	10	58	7,200	4.60	72.14	0	1,811.48
Avg	8.05	502.41	9.75	28.87	5,230.16	4.60	42.04	0	837

Table 6 Performance comparison between Algorithm 1 and B&B on pure integer and mixed integer instances with $p = 2$

	Pure integer					Mixed integer			
	Algorithm 1		B&B			Algorithm 1		B&B	
$m \times n$	**#N**	**T(sec.)**	**#S**	**%G**	**T(sec.)**	**#N**	**T(sec.)**	**%G**	**T(sec.)**
200×100	7.20	2.45	10	0	17.71	5	1.09	0	2.32
200×200	7	7.42	10	0	459.70	5.20	2.51	0	6.07
200×300	6.40	10.29	10	0	378.83	4.60	3.49	0	9.88
200×400	6.80	17.36	10	0	1,973.02	4.60	4.77	0	15.79
Avg	6.85	9.38	10	0	707.31	4.85	2.97	0	8.52
400×200	5.80	11.83	10	0	115.91	4.60	5.51	0	10.97
400×400	7.20	51.10	10	3	1,220.47	4.80	12.59	0	25.08
400×600	6.80	80.70	10	3	1,772.22	4.60	20.07	0	47.78
400×800	7.40	145.43	10	1	3,917.70	4.20	25.87	0	61.76
Avg	6.80	72.27	10	1.75	1,756.58	4.55	16.01	0	36.40
600×300	8.40	50.96	10	0	994.18	4.20	13.02	0	20.14
600×600	5.80	145.57	10	9	2,912.02	4.60	34.46	0	102.28
600×900	6.80	283.66	10	2	5,777.41	4.80	63.91	0	125.36
600×1200	8	641.92	10	20	7,200	4.60	84.89	0	204.25
Avg	7.25	280.53	10	7.75	4,220.90	4.55	49.07	0	113.01
800×400	6.80	90.88	10	0	31.73	4.20	29.66	0	84.177
800×800	11.40	684.77	10	21	6,486.08	4.60	76.56	0	267.462
800×1200	10.20	1,071.83	10	43	7,200	4.40	133.84	0	371.819
800×1600	9.20	1,764.42	8	56	7,200	4.40	195.79	0	405.548
Avg	9.40	902.97	9.50	28.63	5,125.73	4.40	108.96	0	282.25

Appendix B: Algorithm 1 vs SOCP when $p = 3$

A detailed performance comparison between our algorithm and SOCP on the instances with $p = 3$ can be found in Tables 7 and 8 when the convergence tolerance of SOCP is on its default value, i.e., 10^{-12}.

Table 7 Performance comparison between Algorithm 1 and SOCP on pure binary and mixed binary instances with $p = 3$

	Pure binary					Mixed binary			
	Algorithm 1		SOCP			Algorithm 1		SOCP	
$m \times n$	**#N**	**T(sec.)**	**#S**	**%G**	**T(sec.)**	**#N**	**T(sec.)**	**%G**	**T(sec.)**
200×100	19.20	2.57	7	0	10.13	11.70	1.34	0	2.34
200×200	17.10	6.82	7	0	38.29	12	2.39	0	5.53
200×300	15.60	13.24	7	0	87.59	12.60	3.77	0	10.58
200×400	18	28.84	8	0	219.55	12.30	5.10	0	14.76
Avg	17.48	12.87	7.25	0	93.39	12.15	3.15	0	8.30
400×200	16.40	16.48	2	0	53.66	11.40	6.44	0	14.84
400×400	13.80	43.54	2	0	93.20	11.40	11.98	0	37.98
400×600	16.50	177.93	1	0	330.82	11.10	18.52	0	62.40
400×800	15.30	129.70	4	0	967.47	10.20	23.47	0	90.17
Avg	15.50	91.92	2.25	0	499.38	11.03	15.10	0	51.35
600×300	21.30	64.60	0	-	-	11.40	15.79	0	43.69
600×600	14.10	341.89	0	-	-	10.50	29.86	0	132.32
600×900	12.50	1,829.57	0	-	-	10.50	49.83	0	212.06
600×1200	14.10	1,611.47	2	61	7,200	10.50	65.24	0	359.56
Avg	15.50	961.88	0.50	61	7,200	10.73	40.18	0	186.91
800×400	15	61.13	2	0	538.43	11.70	37.72	0	135.12
800×800	15	571.09	0	-	-	12	75.93	0	284.44
800×1200	10.60	1,534.63	5	69	7,200	10.50	111.03	0	485.98
800×1600	12.60	904.61	7	78	7,200	9.90	147.69	0	917.01
Avg	13.30	767.87	3.50	63.64	6,248.34	11.03	93.09	0	455.63

Table 8 Performance comparison between Algorithm 1 and SOCP on pure integer and mixed integer instances with $p = 3$

	Pure integer					Mixed integer			
	Algorithm 1		SOCP			Algorithm 1		SOCP	
$m \times n$	**#N**	**T(sec.)**	**#S**	**%G**	**T(sec.)**	**#N**	**T(sec.)**	**%G**	**T(sec.)**
200×100	18.30	5.94	3	0	12.18	15.60	3.37	0	2.66
200×200	17.70	15.38	3	0	31.49	12.60	5.63	0	7.54
200×300	16.50	23.57	7	0	87.47	12.90	9.24	0	11.80
200×400	17.10	57.27	8	0	163.80	10.80	10.48	0	14.43
Avg	17.40	25.54	5.25	0	97.79	12.98	7.18	0	9.11
400×200	15.30	28.22	1	0	40.53	11.70	12.92	0	16.19
400×400	18.90	170.47	2	0	187.09	12.60	31.58	0	42.72
400×600	16.80	808.42	1	0	425.47	12.60	51.05	0	70.57
400×800	13.50	240.03	1	0	936.87	10.50	59.11	0	95.09
Avg	16.13	311.78	1.25	0	355.41	11.85	38.67	0	56.14
600×300	17.40	110.51	0	-	-	11.70	35.75	0	51.73
600×600	15.30	445.30	1	0	480.87	12.30	91.49	0	159.76
600×900	11.70	725.84	0	-	-	10.50	133.61	0	268.54
600×1200	15.30	1,612.40	4	65	7,200	11.10	191.46	0	320.96
Avg	14.93	723.51	1.25	13	5,856.17	11.40	113.08	0	200.24
800×400	15	209.66	1	0	372.01	11.40	79.60	0	110.55
800×800	15.90	815.31	0	-	-	10.80	188.62	0	347.45
800×1200	13.50	1,338.27	3	72	7,200	11.40	335.26	0	541.40
800×1600	11.40	2,627.90	6	70	7,200	10.80	454.43	0	879.86
Avg	13.95	1,247.79	2.50	63.6	6,517.2	11.10	264.48	0	469.81

A detailed performance comparison between our algorithm and SOCP with the convergence tolerance of 10^{-6} on the pure binary and integer instances with $p = 3$ can be found in Table 9.

Table 9 Performance comparison between Algorithm 1 and SOCP on instances with $p = 3$ for the convergence tolerance of 10^{-6}

	Pure binary					Pure integer				
	Algorithm 1		SOCP			Algorithm 1		SOCP		
$m \times n$	**#N**	**T(sec.)**	**#S**	**%G**	**T(sec.)**	**#N**	**T(sec.)**	**#S**	**%G**	**T(sec.)**
200×100	19.20	2.57	3	0	9.47	18.30	5.94	6	0	7.93
200×200	17.10	6.82	8	0	33.15	17.70	15.38	6	0	30.02
200×300	15.60	13.24	7	0	74.65	16.50	23.57	9	0	78.35
200×400	18	28.84	4	0	182.92	17.10	57.27	9	0	147.17
Avg	17.48	12.87	5.50	0	70.36	17.40	25.54	7.50	0	75.25
400×200	16.40	16.48	2	0	44.36	15.30	28.22	2	0	52.60
400×400	13.80	43.54	2	0	91.16	18.90	170.47	1	0	139.75
400×600	16.50	177.93	1	0	297.88	16.80	808.42	0	-	-
400×800	15.30	129.70	3	0	807.54	13.50	240.03	0	-	-
Avg	15.50	91.92	2	0	373.94	16.13	311.78	0.75	0	81.65
600×300	21.30	64.60	1	0	310.20	17.40	110.51	0	-	-
600×600	14.10	341.89	0	-	-	15.30	445.30	1	0	460.33
600×900	12.50	1,829.57	0	-	-	11.70	725.84	1	53	7,200
600×1200	14.10	1,611.47	2	60	7,200	15.30	1,612.40	3	64	7,200
Avg	15.50	961.88	0.75	20	4,903.4	14.93	723.51	1.25	49	5,852.07
800×400	15	61.13	3	0	528.88	15	209.66	1	0	353.46
800×800	15	571.09	0	-	-	15.90	815.31	0	-	-
800×1200	10.60	1,534.63	4	64	7,200	13.50	1,338.27	4	71	7,200
800×1600	12.60	904.61	9	75	7,200	11.40	2,627.90	8	72	7,200
Avg	13.30	767.87	4	58.19	5,949.17	13.95	1,247.79	3.25	66.15	6,673.34

Appendix C: Algorithm 1 vs SOCP when $p = 4$

A detailed performance comparison between our algorithm and SOCP on the instances with $p = 4$ can be found in Tables 10 and 11 when the convergence tolerance of SOCP is on its default value, i.e., 10^{-12}. A detailed performance comparison between our algorithm and SOCP with the

Table 10 Performance comparison between Algorithm 1 and SOCP on pure binary and mixed binary instances with $p = 4$

	Pure binary						Mixed binary			
	Algorithm 1			SOCP			Algorithm 1		SOCP	
$m \times n$	**#N**	**%G**	**T(sec.)**	**#S**	**%G**	**T(sec.)**	**#N**	**T(sec.)**	**%G**	**T(sec.)**
200×100	15.60	0	4.90	1	0	7.52	30.80	2.96	0	3.09
200×200	12.60	0	10.77	2	0	24.19	27.20	4.80	0	6.23
200×300	12.90	0	35.89	3	0	60.68	27.60	7.80	0	12.28
200×400	10.80	0	85.49	3	0	117.06	22	8.50	0	18.27
Avg	12.98	0	34.26	2.25	0	65.79	26.90	6.02	0	9.97
400×200	11.70	0	26.92	0	-	-	26	13.63	0	18.03
400×400	12.60	0	443.29	0	-	-	28.80	29.57	0	53.02
400×600	12.60	0	178.64	0	-	-	20	33.40	0	102.24
400×800	10.50	0	845.93	0	-	-	20.40	45.06	0	84.54
Avg	11.85	0	373.70	0	-	-	23.80	30.41	0	64.46
600×300	11.70	0	177.38	0	-	-	24.40	33.19	0	64.13
600×600	12.30	0	648.58	0	-	-	19.60	54.95	0	130.47
600×900	10.50	0	415.80	0	-	-	21.20	98.67	0	222.37
600×1200	11.10	0	1,779.17	0	-	-	21.60	135.58	0	429.50
Avg	11.40	0	755.23	0	-	-	21.70	80.60	0	211.62
800×400	11.40	0	168.92	0	-	-	18.80	60	0	136.90
800×800	10.80	0	503.96	0	-	-	18.40	117.05	0	331.42
800×1200	11.40	0	803.93	0	-	-	20	212.15	0	632.41
800×1600	10.80	0	2,624.57	6	85	7,200	18	264.62	0	1,065.65
Avg	11.10	0	1,025.35	1.50	85	7,200	18.80	163.46	0	541.59

Table 11 Performance comparison between Algorithm 1 and SOCP on pure integer and mixed integer instances with $p = 4$

	Pure integer						Mixed integer			
	Algorithm 1			SOCP			Algorithm 1		SOCP	
$m \times n$	**#N**	**%G**	**T(sec.)**	**#S**	**%G**	**T(sec.)**	**#N**	**T(sec.)**	**%G**	**T(sec.)**
200×100	32.70	0	11.20	1	0	8.92	25.20	4.55	0	3.10
200×200	41.20	0	37.91	1	0	26.23	24.40	10.96	0	6.64
200×300	35.60	0	250.81	3	0	57.30	23.20	17.23	0	10.85
200×400	28	0	66.93	4	0	150.77	21.60	20.90	0	17.13
Avg	34.38	0	91.71	2.25	0	90.01	23.60	13.41	0	9.43
400×200	30.80	0	58.30	0	-	-	28.80	32.96	0	19.66
400×400	56	0	879.93	0	-	-	23.20	60.35	0	38.58
400×600	40	0	557.63	0	-	-	19.60	80.85	0	78.88
400×800	36.80	0	595.24	0	-	-	21.20	124.73	0	118.59
Avg	40.90	0	522.77	0	-	-	23.20	74.73	0	63.93
600×300	32	0	214.37	0	-	-	20.80	66.21	0	50.38
600×600	26.80	0	540.55	0	-	-	24	187.22	0	165.47
600×900	32.40	1	1,837.98	0	-	-	21.20	268.38	0	331.15
600×1200	19.20	0	953.24	0	-	-	19.20	353.33	0	371.62
Avg	27.60	0.25	886.53	0	-	-	21.30	218.79	0	229.66
800×400	37.60	0	487.38	0	-	-	18.80	140.61	0	136.76
800×800	24.40	0	1,002.58	0	-	-	20.40	371.33	0	356.37
800×1200	26.20	1.1	2,757.03	0	-	-	19.20	578.22	0	591.10
800×1600	24.10	0.6	3,482.28	3	84	7,200	18.80	824.59	0	1,099.42
Avg	28.08	0.42	1,932.32	0.75	84	7,200	19.30	478.69	0	545.91

convergence tolerance of 10^{-6} on the pure binary and integer instances with $p=4$ can be found in Table 12.

Table 12 Performance comparison between Algorithm 1 and SOCP on instances with $p=4$ for the convergence tolerance of 10^{-6}

	Pure binary						Pure integer					
	Algorithm 1			SOCP			Algorithm 1			SOCP		
$m \times n$	**#N**	**%G**	**T(sec.)**	**#S**	**%G**	**T(sec.)**	**#N**	**%G**	**T(sec.)**	**#S**	**%G**	**T(sec.)**
200×100	15.60	0	4.90	1	0	4.49	32.70	0	11.20	1	0	4.65
200×200	12.60	0	10.77	2	0	23.70	41.20	0	37.91	1	0	21.61
200×300	12.90	0	35.89	2	0	73.26	35.60	0	250.81	2	0	44.59
200×400	10.80	0	85.49	2	0	113.27	28	0	66.93	3	0	89.97
Avg	12.98	0	34.26	1.75	0	60.7	34.38	0	91.71	1.75	0	55.05
400×200	11.70	0	26.92	0	-	-	30.80	0	58.30	0	-	-
400×400	12.60	0	443.29	0	-	-	56	0	879.93	0	-	-
400×600	12.60	0	178.64	1	0	196.93	40	0	557.63	0	-	-
400×800	10.50	0	845.93	0	-	-	36.80	0	595.24	0	-	-
Avg	11.85	0	373.70	0.25	0	196.93	40.90	0	522.77	0	-	-
600×300	11.70	0	177.38	0	-	-	32	0	214.37	0	-	-
600×600	12.30	0	648.58	0	-	-	26.80	0	540.55	0	-	-
600×900	10.50	0	415.80	0	-	-	32.40	1	1,837.98	0	-	-
600×1200	11.10	0	1,779.17	0	-	-	19.20	0	953.24	0	-	-
Avg	11.40	0	755.23	0	-	-	27.60	0.25	886.53	0	-	-
800×400	11.40	0	168.92	0	-	-	37.60	0	487.38	0	-	-
800×800	10.80	0	503.96	0	-	-	24.40	0	1,002.58	0	-	-
800×1200	11.40	0	803.93	1	79	7,200	26.20	1.1	2,757.03	0	-	-
800×1600	10.80	0	2,624.57	3	83	7,200	24.10	0.6	3,482.28	2	86	7,200
Avg	11.10	0	1,025.35	1	82	7,200	28.08	0.42	1,932.32	0.50	86	7,200

References

Ardakan MA, Hamadani AZ (2014) Reliability optimization of seriesparallel systems with mixed redundancy strategy in subsystems. *Reliability Engineering & System Safety* 130:132 – 139, ISSN 0951-8320.

Ben-Tal A, Nemirovski A (2001) On polyhedral approximations of the second-order cone. *Mathematics of Operations Research* 26(2):193–205.

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

Boland N, Charkhgard H, Savelsbergh M (2015) 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.

Bozkurt B, Fowler JW, Gel ES, Kim B, Köksalan M, Wallenius J (2010) Quantitative comparison of approximate solution sets for multicriteria optimization problems with weighted tchebycheff preference function. *Operations Research* 58(3):650–659.

Calkin DE, Montgomery CA, Schumaker NH, Polasky S, Arthur JL, Nalle DJ (2002) Developing a production possibility set of wildlife species persistence and timber harvest value. *Canadian Journal of Forest Research* 32(8):1329–1342.

Chakrabarty D, Devanur N, Vazirani VV (2006) New results on rationality and strongly polynomial time solvability in Eisenberg-Gale markets. *Internet and Network Economics*, volume 4286 of *Lecture Notes in Computer Science*, 239–250 (Springer Berlin Heidelberg).

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.

Coit DW (2001) Cold-standby redundancy optimization for nonrepairable systems. *IIE Transactions* 33(6):471–478, ISSN 1573-9724.

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

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.

Eisenberg E, Gale D (1959) Consensus of subjective probabilities: The pari-mutuel method. *The Annals of Mathematical Statistics* 30(1):165–168.

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.

Feizabadi M, Jahromi AE (2017) A new model for reliability optimization of series-parallel systems with non-homogeneous components. *Reliability Engineering & System Safety* 157:101 – 112, ISSN 0951-8320.

Fischetti M, Glover F, Lodi A (2005) The feasibility pump. *Mathematical Programming* 104(1):91–104, ISSN 1436-4646.

Fischetti M, Lodi A (2003) Local branching. *Mathematical Programming* 98(1):23–47, ISSN 1436-4646.

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

Grötschel M, Lovasz L, Schrijver A (1988) *Geometric Algorithms and Combinatorial Optimization* (Berlin: Springer-Verlag).

Haider Z, Charkhgard H, Kwon C (2018) A robust optimization approach for solving problems in conservation planning. *Ecological Modelling* 368:288 – 297.

Hamming RW (1950) Error detecting and error correcting codes. *Bell System Technical Journal* 29(2):147–160.

Hooker J (2011) *Logic-based methods for optimization: combining optimization and constraint satisfaction*, volume 2 (John Wiley & Sons).

IBM ILOG CPLEX Optimization Studio (2016) CPLEX Parameters Reference. URL `https://goo.gl/Fv5R2Z`.

Jain K, Vazirani VV (2007) Eisenberg-Gale markets: Algorithms and structural properties. *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, 364–373, STOC '07 (New York, NY, USA: ACM).

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

Karasakal E, Köksalan M (2009) Generating a representative subset of the nondominated frontier in multiple criteria decision making. *Operations Research* 57(1):187–199.

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.

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

Masin M, Bukchin Y (2008) Diversity maximization approach for multiobjective optimization. *Operations Research* 56(2):411–424.

Nash JF (1950) The bargaining problem. *Econometrica* 18:155–162.

Nash JF (1953) Two-person cooperative games. *Econometrica* 21:128–140.

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

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

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

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

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

Ryoo HS, Sahinidis NV (2003) Global optimization of multiplicative programs. *Journal of Global Optimization* 26(4):387–418.

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.

Sayın S (2003) A procedure to find discrete representations of the efficient set with specified coverage errors. *Operations Research* 51(3):427–436.

Serrano R (2005) Fifty years of the Nash program 1953-2003. *Investigaciones Economicas* 219–258.

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.

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

Vazirani VV (2012) Rational convex programs and efficient algorithms for 2-player Nash and nonsymmetric bargaining games. *SIAM J. discrete math* 26(3):896–918.

Williams PH, Araújo MB (2002) Apples, oranges, and probabilities: Integrating multiple factors into biodiversity conservation with consistency. *Environmental Modeling & Assessment* 7(2):139–151.