Exact Solution Approaches for Integer Linear Generalized Maximum Multiplicative Programs Through the Lens of Multi-objective Optimization

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

We study a class of single-objective nonlinear optimization problems, the so-called Integer Linear Generalized Maximum Multiplicative Programs (IL-GMMP). This class of optimization problems has a significant number of applications in different fields of study including but not limited to game theory, systems reliability, and conservative planning. An IL-GMMP can be reformulated as a mixed integer Second-Order Cone Program (SOCP) and therefore, can be solved effectively by commercial solvers such as IBM ILOG CPLEX, Gurobi, and FICO Xpress. In this study, we show that IL-GMMPs can be viewed as special cases of the problem of optimization over the efficient (or Pareto-optimal) set in multi-objective integer linear programming. Based on this observation, we develop three exact solution approaches with a desirable property: they only solve a number of single-objective integer linear programs to compute an optimal solution of an IL-GMMP. Through an extensive computational study with 57600 experiments, we compare the performance of all three algorithms using the three main commercial single-objective integer linear programming solvers in the market: CPLEX, Gurobi, and Xpress. We also compare the performance of our algorithms using the mixed integer SOCP solvers of CPLEX, Gurobi, and Xpress. The results show that the choice of a commercial solver impacts the solution time dramatically and that, by the right choice of solver, one of our proposed algorithms is significantly faster than other methods. We also illustrate that although it is possible to linearize IL-GMMPs, commercial solvers struggle to solve such linearized instances.

Key words: Multi-objective integer linear programming, Generalized maximum multiplicative programming, Criterion space search algorithms, Nash bargaining solution, Geometric mean optimization

1. Introduction

A Generalized Maximum Multiplicative Program (GMMP) is the problem of the form,

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

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 and $\boldsymbol{\pi}:=\big(\pi_1,\cdots,\pi_p\big)$ is the so-called vector of *geometric weights* where $\pi_i>0$ for all $i=1,\ldots,p$. As an aside, throughout this article,

vectors are always column-vectors and are denoted in bold fonts. It is assumed that the vector of geometric weights is known and the optimal objective value of a GMMP is strictly positive. It is worth mentioning that, in this study, the term 'Generalized' in GMMP refers to the fact that the vector of geometric weights is not necessarily a unit vector, i.e., geometric weights are not necessarily equal to one. We note that if $\mathcal{X}$ is defined by a set of linear constraints, the problem is referred to as Linear GMMP (L-GMMP). If in a L-GMMP, all decision variables are integer, the problem is referred to as Integer Linear GMMP (IL-GMMP). Finally, if in a L-GMMP, some but not all decision variables are integer, the problem is referred to as Mixed Integer Linear GMMP (MIL-GMMP).

The goal of this study is to develop new exact and fast algorithms that can solve any IL-GMMP by just solving a number of single-objective integer linear programs. In other words, we attempt to develop new algorithms that can exploit the power of commercial single-objective integer linear programming solvers such as IBM ILOG CPLEX, Gurobi, and FICO Xpress for solving an IL-GMMP which is a non-linear optimization problem. At the end of this paper (see Section 6), we will explain how our proposed algorithms can be easily modified to solve any MIL-GMMP. The reason that we do not focus on MIL-GMMPs directly in this study is that in our proposed modification technique, some non-linear convex optimization problems should be solved during the course of our algorithms and this is not compatible with our goal in this paper, i.e., solving only (integer) linear programs.

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,

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

Specifically, by maximizing the function $\prod_{i=1}^{p} y_i^{\pi_i}(\boldsymbol{x})$ 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 Charkhgard et al. (2018), Saghand et al. (2019), Saghand and Charkhgard (2019), and Nash (1950, 1953). 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, Boland et al. 2017, Sayın 2000, Benson 1984).

1.1. Motivation

The main motivation of this study is based on this observation that GMMPs have many applications in different fields of study. Consequently, developing faster and more reliable algorithms for this class of optimization problems can have a significant impact on problem solving in many fields. Next, we review some of the well-known applications.

The first field of study is game theory. One of the important applications of GMMPs is finding the Nash solution to a symmetric/non-symmetric bargaining problem. A bargaining problem is a cooperative game where 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 bargaining problem was proposed by Nash and is now known as the Nash bargaining solution (Nash 1950, 1953). Nash proved that, under certain conditions, an optimal solution of the bargaining problem can be obtained by solving a GMMP in which $y_i(\boldsymbol{x})$ represents the utility function of player $i \in \{1, \ldots, p\}$ for a feasible solution. It is worth mentioning that GMMPs have other applications in game theory as well, e.g., computing a market equilibrium for a linear fisher market or a Kelly capacity allocation market. Interested readers can refer to Charkhgard et al. (2018), Chakrabarty et al. (2006), Eisenberg and Gale (1959), Jain and Vazirani (2007), and Vazirani (2012) for further details about other applications.

The second field is multi-objective optimization. As mentioned earlier, a natural application of GMMPs is in multi-objective optimization based on the reason that they can be viewed as special cases of the problem of optimization over the efficient set. Multi-objective optimization is a critical tool in management, where competing goals must often be considered and balanced when making decisions (Stidsen et al. 2014, Özpeynirci and Köksalan 2010, Phelps and Köksalan 2003, Erlebach et al. 2002, Johnson and Hurter 2000, Sayın and Kouvelis 2005, Engau and Wiecek 2008, Wallenius et al. 2008). For example, in business settings, there may be trade-offs between long-term profits and short-term cash flows or between cost and reliability, while in public good settings, there can be trade-offs between providing benefits to different communities or between environmental impacts and social good. Therefore, computing the set of efficient solutions in a multi-objective optimization problem can be highly valuable. However, it is known 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. 2017). 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 optimization over the efficient set, which is a global optimization problem (Benson 1984).

The third field is conservation planning. A typical conservation planning problem is about 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. A typical approach for preserving biodiversity is to use GMMPs in which $y_i(\boldsymbol{x})$ is the survival probability of species $i \in \{1, \ldots, p\}$ for a feasible solution $\boldsymbol{x} \in \mathcal{X}$ (Calkin et al. 2002, Nicholson and Possingham 2006b, Williams and Araújo 2002).

Finally, it is worth mentioning that there are also other fields of study that give rise to GMMPs such as system reliability and maximum likelihood estimation (Coit 2001, Ardakan and Hamadani 2014, Feizabadi and Jahromi 2017, Charkhgard et al. 2018). For example, maximizing the reliability of series-parallel systems can be done by using GMMPs in which $y_i(\boldsymbol{x})$ is the reliability of subsystem $i \in \{1, \ldots, p\}$ for a feasible solution $\boldsymbol{x} \in \mathcal{X}$.

1.2. Background and contributions

Due to the importance of GMMPs, several exact approaches exist for solving different subclasses of GMMPs in the relevant literature. However, the underlying assumption of (almost) all existing approaches is that all geometric weights are equal to one, i.e., $\pi_i = 1$ for all $i = 1, \ldots, p$. Of course, it is known that any GMMP can be transformed into a GMMP with only unit weights by introducing some new variables and constraints (Kalai 1977). Specifically, we need to first change π_i to a positive integer number for all $i = 1, \ldots, p$ by multiplying all of them to a sufficiently large positive number. Afterwards, if there exists $\pi_i > 1$ for some $i \in \{1, \ldots, p\}$ then $\pi_i - 1$ copies of y_i should be created and added to the formulation. We observe that such a transformation can undesirably increase the size of the formulation significantly. Hence, one of the main contributions of this study is that our proposed approaches directly deal with geometric weights without changing the size of the formulation. Next, we review some of the existing approaches.

For L-GMMPs, a typical solution approach is to use a log-transformation of the objective function, i.e., $\sum_{i=1}^{p} \pi_i \log y_i(\boldsymbol{x})$, and then solving the transformed problem (in polynomial time) using a convex programming solver (Grötschel et al. 1988). However, Charkhgard et al. (2018) showed that a faster approach for solving a L-GMMP is to transform the problem into a Second-Order Cone Program (SOCP) using the technique introduced by (Ben-Tal and Nemirovski 2001) and then solving it using commercial solvers such as IBM ILOG CPLEX, Gurobi, or FICO Xpress. Since linear programming solvers are significantly faster than convex programming solvers (in general), some authors have recently shown that even faster algorithms can be developed for solving L-GMMPs which only solve a number of linear programs (Charkhgard et al. 2018, Vazirani 2012).

For IL-GMMPs and MIL-GMMPs, an effective approach is to transform the problem into a mixed integer SOCP and then solving the problem using IBM ILOG CPLEX, Gurobi, or FICO Xpress (**?**Saghand and Charkhgard 2019). However, for MIL-GMMPs with $p = 2$, **?** proposed a faster algorithm by incorporating the linear programming based approach developed by Charkhgard et al. (2018) for L-GMMPs in an effective branch-and-bound framework. Recently, Saghand and Charkhgard (2019) proposed an even more effective approach for solving MIL-GMMPs with $p \geq 2$ which solves a number of single-objective integer linear programs and SOCPs to compute an optimal solution. As an aside, we note that, in this study, one of the algorithms that we have developed is a significantly modified/enhanced version of the algorithm proposed by Saghand and Charkhgard (2019). We show that the modified algorithm significantly outperforms the one proposed by Saghand and Charkhgard (2019) for solving instances of IL-GMMPs because of the new bounding and cut generation techniques. Moreover, another advantage of the modified algorithm is that, in contrast to the algorithm proposed by Saghand and Charkhgard (2019), it can directly deal with the non-unit geometric weights.

In light of the above, the main contributions of this study are as follows:

- Developing three new exact algorithms for solving any IL-GMMP with two desirable characteristics: they only solve a number of single-objective integer linear programs to compute an optimal solution and also they directly deal with (non-unit) geometric weights.
- Developing a novel procedure for computing dual bounds for IL-GMMPs (which also works for MIL-GMMPs) and incorporating it in our proposed exact algorithms. The proposed procedure can compute a dual bound from a feasible solution in linear time if it is generated in a specific way. This procedure has not been employed in any existing methods (to the best of our knowledge).
- Developing an effective cut, the so-called hypotenuse cut, which can be added to the problem using any feasible solution with strictly positive objective values, and incorporating it in our proposed exact algorithms.
- Conducting an extensive computational study with 57600 experiments in which for the first time (to the best of our knowledge), the performance of three main commercial mixed integer SOCP solvers, i.e., CPLEX, Gurobi, and FICO Xpress, are compared on solving IL-GMMPs. Overall, we show that for our test instances, Xpress is the most reliable commercial mixed integer SOCP solver and is faster than the other commercial solvers by a factor of at least 10 on many instances. We also compare the performance of our three proposed algorithms under different single-objective integer programming solvers, i.e., CPLEX, Gurobi, and Xpress. Overall, we show that for our test instances, single-objective integer programming solver of CPLEX aligns significantly better with our proposed algorithms. All our proposed methods with CPLEX are competitive with each other but one of them can even outperform the best mixed integer SOCP solver by a factor of at least 2

on many instances. Furthermore, we show that linearizing (the objective function of) IL-GMMPs can result in large single-objective integer linear programs which commercial solvers struggle to solve. Specifically, our proposed techniques can solve our largest instances in less than 900 seconds while commercial solvers cannot even find a feasible solution for the smallest linearized instances within an hour.

1.3. Structure

The remainder of this paper is organized as follows. In Section 2, we give preliminaries. In Section 3, we introduce a high-level description of our proposed algorithms and also their key operations. In Section 4, the detailed descriptions of the proposed algorithms are given. In Section 5, an extensive computational study is provided. Finally, in Section 6, some concluding remarks are given.

2. Preliminaries

An IL-GMMP can be stated as follows,

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

where n_b and n_i represent the number of binary and integer decision variables, respectively, D is a $p \times n$ matrix where $n := n_b + n_i$, $\boldsymbol{d}$ is a p-vector, A is a $m \times n$ matrix, and $\boldsymbol{b}$ is a m-vector. For notational convenience, we partition the index set of variables $\mathcal{N} := \{1, 2, ..., n\}$ into binary $\mathcal{B}$ and integer $\mathcal{I}$. We note that in this study, it is assumed that $\pi_i > 0$ and it is not necessarily integer for all $i = 1, \ldots, p$.

In this paper, the sets $\mathcal{X} := \{\boldsymbol{x} \in \mathbb{B}^{n_b} \times \mathbb{Z}^{n_i} : A\boldsymbol{x} \leq \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0}\}$ and $\mathcal{Y} := \{\boldsymbol{y} \in \mathbb{R}^p : \boldsymbol{x} \in \mathcal{X},\ \boldsymbol{y} = D\boldsymbol{x} + \boldsymbol{d},\ \boldsymbol{y} \geq \boldsymbol{0}\}$ represent 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 that 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 Section 1, an IL-GMMP can be reformulated as a mixed integer SOCP if π_i is a positive integer for all $i = 1, \ldots, p$ (Ben-Tal and Nemirovski 2001). 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 (3) can be reformulated as follows,

$$\max \left\{ \gamma :\ 0 \leq \gamma \leq \left(\prod_{i=1}^{p} y_i^{\pi_i} \right)^{\frac{1}{\sum_{i=1}^{p} \pi_i}},\ \boldsymbol{y} \in \mathcal{Y} \right\}.$$

It is evident that if $\bar{\gamma}$ is the optimal objective value of the reformulated problem, then $\bar{\gamma}^{\sum_{i=1}^{p}\pi_i}$ is the optimal objective value of Problem (3). Let k be the smallest integer value such that $2^k \geq \sum_{i=1}^{p}\pi_i$. By introducing a set of non-negative variables and constraints, the geometric mean constraint can be replaced as follows:

$$
\begin{aligned}
&\max \ \gamma \\
&\text{s.t. } 0 \leq \gamma \leq \sqrt{\tau_1^{k-1}\tau_2^{k-1}} \\
&\qquad 0 \leq \tau_j^l \leq \sqrt{\tau_{2j-1}^{l-1}\tau_{2j}^{l-1}} && \text{for } j = 1,\ldots,2^{k-l} \text{ and } l = 1,\ldots,k-1 \\
&\qquad 0 \leq \tau_j^0 = y_i && \text{for } j = \big(\sum_{k=1}^{i-1}\pi_k\big)+1,\ldots,\big(\sum_{k=1}^{i}\pi_k\big) \text{ and } i = 1,\ldots,p \\
&\qquad 0 \leq \tau_j^0 = \gamma && \text{for } j = \big(\sum_{k=1}^{p}\pi_k\big)+1,\ldots,2^k \\
&\qquad \boldsymbol{y} \in \mathcal{Y}.
\end{aligned}
$$

The above formulation is 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 inter SOCP solver such as IBM ILOG CPLEX, Gurobi, and FICO Xpress, to solve any IL-GMMP. However, the goal of this paper is to develop effective solution approaches based on the following definition and theoretical results.

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

$$
\begin{aligned}
&y_i \leq y_i' && \forall i \in \{1,2,...,p\} \\
&y_i < y_i' && \text{for at least one } i \in \{1,2,...,p\}
\end{aligned}
$$

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

Proof. Suppose that $\boldsymbol{x}^*$ is an optimal solution of Problem (3) 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

$$
\begin{aligned}
&y_i^* \leq y_i && \forall i \in \{1,2,...,p\} \\
&y_i^* < y_i && \text{for at least one } i \in \{1,2,...,p\}
\end{aligned}
$$

where $\boldsymbol{y} := D\boldsymbol{x} + \boldsymbol{d}$. In addition, by assumptions of Problem (3), we know that $\pi_i > 0$ for all $i = 1, \cdots, p$ and $\boldsymbol{y}^* > \boldsymbol{0}$. Therefore, we must have that $0 < \prod_{i=1}^{p} {y^*}_i^{\pi_i} < \prod_{i=1}^{p} y_i^{\pi_i}$. Consequently, $\boldsymbol{x}^*$ cannot be an optimal solution (a contradiction). Q.E.D.

Proposition 1 indicates that

$$\max_{\boldsymbol{y} \in \mathcal{Y}} \prod_{i=1}^{p} y_i^{\pi_i} = \max_{\boldsymbol{y} \in \mathcal{Y}_N} \prod_{i=1}^{p} y_i^{\pi_i}$$

Therefore, as we discussed earlier in Section 1, an IL-GMMP is a special case of the problem of optimization over the efficient set. Hence, instead of searching the entire feasible set, we propose algorithms which look for an optimal point of Problem (3) by searching over the set of nondominated points. We next present a critical proposition and corollary motivated by the study of Charkhgard et al. (2018) that result in generating effective cuts during the course of our proposed algorithms.

PROPOSITION 2. *If $\bar{\boldsymbol{y}} > \boldsymbol{0}$ and $\boldsymbol{\pi} > \boldsymbol{0}$ then $\bar{\boldsymbol{y}}$ is the unique optimal point of the following L-GMMP,*

$$\max \left\{ \prod_{i=1}^{p} y_i^{\pi_i} : \ \boldsymbol{y} \geq \boldsymbol{0}, \ \sum_{i=1}^{p} \frac{\pi_i}{\bar{y}_i} y_i \leq \sum_{i=1}^{p} \pi_i \right\}.$$

Proof. It is known that every L-GMMP has a unique optimal point (Nash 1953). So, let $\boldsymbol{y}^*$ be the optimal point of the given L-GMMP. In order to find $\boldsymbol{y}^*$, we rewrite the given L-GMMP as the following convex optimization problem,

$$\boldsymbol{y}^* \in \arg\min \left\{ -\sum_{i=1}^{p} \pi_i \log(y_i) : \ \boldsymbol{y} \geq \boldsymbol{0}, \ \sum_{i=1}^{p} \frac{\pi_i}{\bar{y}_i} y_i \leq \sum_{i=1}^{p} \pi_i \right\}.$$

By assuming that $y \in \mathbb{R}^p$ (rather than $\boldsymbol{y} \in \mathbb{R}^p_{\geq}$), we relax the problem to obtain,

$$\boldsymbol{y}^*_R \in \arg\min \{ -\sum_{i=1}^{p} \pi_i \log(y_i) : \ \sum_{i=1}^{p} \frac{\pi_i}{\bar{y}_i} y_i - \sum_{i=1}^{p} \pi_i \leq 0 \}$$

The KKT conditions of the relaxed problem are as follows,

$$\begin{aligned}
\frac{-\pi_i}{y_i} + \eta \frac{\pi_i}{\bar{y}_i} &= 0 && \forall i \in \{1, \ldots, p\} \\
\eta (\sum_{i=1}^{p} \frac{\pi_i}{\bar{y}_i} y_i - \sum_{i=1}^{p} \pi_i) &= 0 \\
\sum_{i=1}^{p} \frac{\pi_i}{\bar{y}_i} y_i - \sum_{i=1}^{p} \pi_i &\leq 0 \\
\eta &\geq 0
\end{aligned}$$

where η is the dual variable associated with constraint $\sum_{i=1}^{p} \frac{\pi_i}{\bar{y}_i} y_i - \sum_{i=1}^{p} \pi_i \leq 0$. It evident that $\boldsymbol{y}^*_R = \bar{\boldsymbol{y}}$ and $\eta = 1$ are feasible for the KKT conditions. Moreover, $\bar{\boldsymbol{y}} > \boldsymbol{0}$, and consequently, $\boldsymbol{y}^* = \bar{\boldsymbol{y}}$. Q.E.D.

As an aside, in Proposition 2, the set $\{\boldsymbol{y} \in \mathbb{R}^p : \boldsymbol{y} \geq \mathbf{0},\ \sum_{i=1}^{p} \frac{\pi_i}{\bar{y}_i} y_i \leq \sum_{i=1}^{p} \pi_i\}$ can be viewed as a right-angled triangle when $p = 2$ and its generalization (for higher dimensions) when $p \geq 2$. So, the proposition suggests that for such a generalized right-angled triangle in the criterion space, the optimal point is always on the hypotenuse and can be computed easily.

COROLLARY 1. *Let $\bar{\boldsymbol{y}} > \mathbf{0}$ and $\boldsymbol{y}^*$ represent a feasible point and the optimal point of an IL-GMMP (with feasible set $\mathcal{Y}$), respectively. If $\mathcal{Y} \subseteq \left\{\boldsymbol{y} \in \mathbb{R}^p : \boldsymbol{y} \geq \mathbf{0},\ \sum_{i=1}^{p} \frac{\pi_i}{\bar{y}_i} y_i \leq \sum_{i=1}^{p} \pi_i\right\}$ then we have that $\boldsymbol{y}^* = \bar{\boldsymbol{y}}$. Otherwise, if $\mathcal{Y} \nsubseteq \left\{\boldsymbol{y} \in \mathbb{R}^p : \boldsymbol{y} \geq \mathbf{0},\ \sum_{i=1}^{p} \frac{\pi_i}{\bar{y}_i} y_i \leq \sum_{i=1}^{p} \pi_i\right\}$ then either $\boldsymbol{y}^* = \bar{\boldsymbol{y}}$ or $\boldsymbol{y}^* \in \left\{\boldsymbol{y} \in \mathcal{Y} : \sum_{i=1}^{p} \frac{\pi_i}{\bar{y}_i} y_i > \sum_{i=1}^{p} \pi_i\right\}$.*

Corollary 1 suggests that, for any strictly positive feasible point $\bar{\boldsymbol{y}} \in \mathcal{Y}$ and $\bar{\boldsymbol{y}} > \mathbf{0}$, the following cut (or half space),

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

can be added to a IL-GMMP because this inequality does not cut off its optimal point. We sometimes refer to this inequality as the *hypotenuse cut*. Note that one can write the proposed cut in the form of strict inequality, i.e., $>$, based on Corollary 1 and that obviously results in a stronger cut but to avoid numerical issues, we use a weaker cut. Also, note that in this paper, we introduced the hypotenuse cut in the context of IL-GMMPs but one can easily infer that the same cut is valid for any GMMP. Next, we present a theorem that plays a significant role when computing a dual bound in our algorithms.

THEOREM 1. *For a set of non-negative real numbers $\{c_1, \ldots, c_p\}$ and a set of positive real numbers $\{\pi_1, \ldots, \pi_p\}$, the following inequality,*

$$c_1^{\pi_1} c_2^{\pi_2} \ldots c_p^{\pi_p} \leq \left(\frac{\sum_{i=1}^{p} \pi_i c_i}{\sum_{i=1}^{p} \pi_i} \right)^{\sum_{i=1}^{p} \pi_i},$$

holds.

Proof. The proof of this theorem is based on the *Arithmetic Mean-Geometric Mean* (AM-GM) inequality (Steele 2004). For a set of non-negative real numbers $\{c_1, \ldots, c_p\}$ and a set of positive real numbers $\{w_1, \ldots, w_p\}$ with $\sum_{i=1}^{p} w_i = 1$, the following inequality,

$$c_1^{w_1} c_2^{w_2} \ldots c_p^{w_p} \leq \sum_{i=1}^{p} w_i c_i,$$

is known as the AM-GM inequality. So, by setting $w_i = \frac{\pi_i}{\sum_{j=1}^{p} \pi_j}$ in the AM-GM inequality for all $i = 1, \ldots, p$, we have that,

$$c_1^{\frac{\pi_1}{\sum_{j=1}^{p} \pi_j}} c_2^{\frac{\pi_2}{\sum_{j=1}^{p} \pi_j}} \ldots c_p^{\frac{\pi_p}{\sum_{j=1}^{p} \pi_j}} \leq \frac{\sum_{i=1}^{p} \pi_i c_i}{\sum_{i=1}^{p} \pi_i}.$$

Hence,

$$\left(c_1^{\pi_1} c_2^{\pi_2} \dots c_p^{\pi_p}\right)^{\frac{1}{\sum_{j=1}^{p} \pi_j}} \leq \frac{\sum_{i=1}^{p} \pi_i c_i}{\sum_{i=1}^{p} \pi_i},$$

and the result follows. Q.E.D.

Theorem 1 implies the following critical remark for computing dual and primal bounds for IL-GMMPs.

REMARK 1. If $\bar{\boldsymbol{y}}$ is an optimal point of the following integer linear program,

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

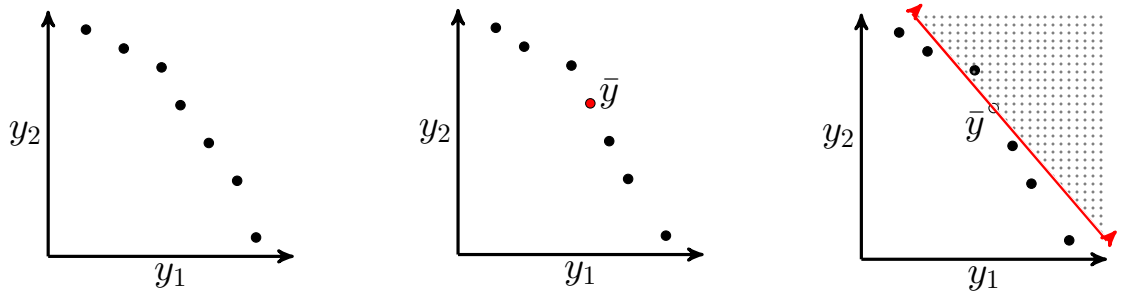
where $\boldsymbol{\pi} > \mathbf{0}$ then the following inequalities,

$$\prod_{i=1}^{p} \bar{y}_i{}^{\pi_i} \leq \max_{\boldsymbol{y} \in \mathcal{Y}} \prod_{i=1}^{p} y_i^{\pi_i} \leq \left(\frac{\sum_{i=1}^{p} \pi_i \bar{y}_i}{\sum_{i=1}^{p} \pi_i}\right)^{\sum_{i=1}^{p} \pi_i},$$

hold.

3. High-level description and key operations

In this section, we first provide a high-level description of our proposed algorithms and then we provide a detailed description of their key operations. First, observe that the nondominated frontier of a multi-objective integer linear program is discrete and finite because we assume that $\mathcal{X}$ is bounded. An illustration of the nondominated frontier when $p = 2$ can be found in Figure 1a.



(a) Nondominated frontier (b) A nondominated point (c) The hypotenuse cut

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

In each iteration, the proposed algorithms attempt to compute a nondominated point (if possible) in the criterion space, i.e., denoted by $\bar{\boldsymbol{y}}$, using a specific operation. This operation is specifically designed based on Remark 1 and its details are given in Subsection 3.2. In other words, $\bar{\boldsymbol{y}}$ results in computing both primal and dual bounds. An illustration of $\bar{\boldsymbol{y}}$ obtained in the first iteration when $p = 2$ can be found in Figure 1b.

Next, the proposed algorithms add the so-called no-good constraint to the formulation to ensure that the obtained feasible solution associated with $\bar{\boldsymbol{y}}$ in the decision space, i.e., $\bar{\boldsymbol{x}}$, will be excluded from the search in the future iterations. The algorithms (if $\bar{\boldsymbol{y}} > \mathbf{0}$) also add a hypotenuse cut based on $\bar{\boldsymbol{y}}$ to the problem to remove the parts of the criterion space that cannot contain points better than $\bar{\boldsymbol{y}}$. An illustration of such a cut can be found in Figure 1c.

(a) Top region (b) Right region

Figure 2 An illustration of a possible search strategy

It is evident that, by repeating the process explained above in the remaining feasible region, one will be able to eventually find an optimal solution of Problem (3). Specifically, in each iteration, the global primal and dual bounds will be updated and the algorithms terminate as soon as the optimality gap falls below a given threshold. Overall, the process explained above is the underlying idea of all our proposed methods. However, they employ different search mechanisms. For example, one algorithm searches the remaining region in Figure 1c directly and another algorithm first decomposes it into two parts (because $p = 2$ in this example) and then searches each one independently. An illustration of such a decomposition can be found in Figure 2 in which the remaining search region in Figure 1c is decomposed into two parts including top and right regions. In Figure 2 the top and right regions are created based on this observation that the dominated region by a nondominated point $\bar{\boldsymbol{y}}$, i.e., $\{\boldsymbol{y} \geq \mathbf{0} : \boldsymbol{y} \leq \bar{\boldsymbol{y}}\}$, can be removed from the search region. Next, the key operations used in our algorithms are explained.

3.1. No-good constraint

As mentioned earlier in this section, no-good constraints, also known as *tabu* constraints (Hooker 2011, Fischetti and Lodi 2003, Hamming 1950), play an important role in our proposed algorithms. It is known that no-good constraints for problems with general binary decision variables are naturally linear, whereas, for problems with general integer decision variables, they are naturally non-linear (Fischetti et al. 2005). Although the linearization of such non-linear constraints is possible, due to the computational complexity that will be added to the problem, it is more

convenient to first convert all the integer variables to binaries and then add the linear no-good constraints to problems. Due to this reason, in the rest of this paper, whenever we want to solve an IL-GMMP by the proposed algorithms, we assume that all variables are binary, i.e., $\mathcal{I} = \emptyset$ and $n = n_b$. Note that since 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,

$$2^0 z_1 + 2^1 z_2 + ... + 2^{v-1} z_v \leq u.$$

With this in mind, in our proposed algorithms, we maintain a list of feasible points and their corresponding solutions, denoted by POOL, found during the course of these algorithms. Specifically, whenever a feasible point $\boldsymbol{y}'$ and its corresponding solution $\boldsymbol{x}'$ is found, $(\boldsymbol{x}', \boldsymbol{y}')$ will be added to POOL. For each $(\boldsymbol{x}', \boldsymbol{y}') \in$ POOL, the proposed algorithms will add the following no-good constraint,

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

where $\mathcal{S}^0 := \{j \in \mathcal{B} : x'_j = 0\}$ and $\mathcal{S}^1 := \{j \in \mathcal{B} : x'_j = 1\}$, to ensure that $\boldsymbol{x}'$ will be excluded from their search in the future iterations.

3.2. Weighted sum operation

Following Proposition 1, any *efficient* solution is expected to be a high-quality feasible solution for Problem (3) and provides a good (global) lower/primal bound for the problem. Therefore, the proposed algorithms attempt to compute a *locally* efficient solution in each iteration. Note that the reason for using the term 'locally' is that in each iteration the algorithms remove a set of feasible solutions from the problem by adding some constraints. With this in mind, the *locally* efficient solution will certainly be a nondominated point for the restricted feasible set but not necessarily for the entire problem.

To obtain a *locally* efficient solution (if possible), we will use a well-known observation in multi-objective optimization (see for instance Ehrgott (2005)): optimizing any positive weighted summation of the objective functions over the feasible set must return an efficient solution (if exists any). In light of this observation, in this research, we simply assume that the weights are shown by the vector $\boldsymbol{w} > \boldsymbol{0}$, and therefore, we maximize $\sum_{i=1}^{p} w_i y_i$ over $y \in \mathcal{Y}$ but with some additional constraints.

Note that the additional constraints include the no-good constraints as well as the hypotenuse cuts. Therefore, the solution returned by solving the weighted sum optimization after adding such constraints (in particular the no-good constraints) is not guaranteed to be a true efficient solution for the corresponding multi-objective problem without 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 weighted sum optimization problem employed by our algorithms, denoted by $\mathrm{WSO}(\boldsymbol{w}, \textsc{Pool}, \boldsymbol{l})$, can be stated as follows,

$$
\begin{aligned}
(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \in \arg\max \Bigg\{ & \sum_{i=1}^{p} w_i y_i : \\
& \boldsymbol{y} \in \mathcal{Y}, \\
& y_i \geq l_i && \forall i \in \{1, \cdots, p\} \\
& \sum_{i=1}^{p} \frac{\pi_i}{y'_i} y_i \geq \sum_{i=1}^{p} \pi_i && \forall (\boldsymbol{x}', \boldsymbol{y}') \in \textsc{Pool} \text{ with } \boldsymbol{y}' > \boldsymbol{0} \\
& \sum_{j \in \mathcal{S}^0(\boldsymbol{x}')} x_j + \sum_{j \in \mathcal{S}^1(\boldsymbol{x}')} (1 - x_j) \geq 1 && \forall (\boldsymbol{x}', \boldsymbol{y}') \in \textsc{Pool} \Bigg\},
\end{aligned}
$$

where $\boldsymbol{l} \geq \boldsymbol{0}$ is the vector of lower bound values for $\boldsymbol{y}$ that defines a search region in the criterion space. Each algorithm in this study updates the vector $\boldsymbol{l}$ according to its underlying search mechanism in each iteration.

Remark 2. Due to Remark 1, after solving $\mathrm{WSO}(\boldsymbol{\pi}, \emptyset, \boldsymbol{0})$, global primal and dual bounds for Problem (3), i.e., $\max_{\boldsymbol{y} \in \mathcal{Y}} \prod_{i=1}^{p} y_i^{\pi_i}$, can be computed in linear time,

$$
\prod_{i=1}^{p} \bar{y_i}^{\pi_i} \leq \max_{\boldsymbol{y} \in \mathcal{Y}} \prod_{i=1}^{p} y_i^{\pi_i} \leq \left(\frac{\sum_{i=1}^{p} \pi_i \bar{y_i}}{\sum_{i=1}^{p} \pi_i} \right)^{\sum_{i=1}^{p} \pi_i}.
$$

Similarly, after solving $\mathrm{WSO}(\boldsymbol{\pi}, \textsc{Pool}, \boldsymbol{l})$, global primal and dual bounds for the restricted problem, i.e., $\max_{\boldsymbol{y} \in \bar{\mathcal{Y}}} \prod_{i=1}^{p} y_i^{\pi_i}$ where $\bar{\mathcal{Y}}$ is the feasible set of $\mathrm{WSO}(\boldsymbol{\pi}, \textsc{Pool}, \boldsymbol{l})$ in the criterion space can be computed in linear time,

$$
\prod_{i=1}^{p} \bar{y_i}^{\pi_i} \leq \max_{\boldsymbol{y} \in \bar{\mathcal{Y}}} \prod_{i=1}^{p} y_i^{\pi_i} \leq \left(\frac{\sum_{i=1}^{p} \pi_i \bar{y_i}}{\sum_{i=1}^{p} \pi_i} \right)^{\sum_{i=1}^{p} \pi_i}.
$$

4. Proposed Algorithms

In this section, we provide a detailed description of our proposed algorithms and the main reasons for developing them. However, before doing so, we would like to mention a helpful remark.

REMARK 3. All our proposed algorithms solve only a number of single-objective integer linear programs to compute an optimal solution of an IL-GMMP. Also, the proposed algorithms in this study are finite due to the following two reasons: (1) in each iteration a feasible solution will be excluded from the feasible region because of adding no-good constraints; and (2) the number of feasible solutions is finite because $\mathcal{X}$ is bounded by assumptions.

4.1. The Criterion Feasible Set Shrinking Algorithm I

The Criterion Feasible Set Shrinking Algorithm I (CFSSA-I) is the simplest but the most effective algorithm (according to our numerical results) that we have developed. The algorithm maintains three pieces of information including: a pool of solutions found during the search denoted by POOL, a global upper/dual bound denoted by $\mathrm{G_{UB}}$, and a global primal/lower bound denoted by $\mathrm{G_{LB}}$. At the beginning, POOL is empty, and we set $\mathrm{G_{UB}} = +\infty$ and $\mathrm{G_{LB}} = -\infty$. The algorithm iteratively shrinks the feasible set in the criterion space until it becomes empty and/or the optimality gap falls below a given threshold. At the end, the algorithm returns the best solution found and its image in the criterion space, denoted by $(\boldsymbol{x}^*, \boldsymbol{y}^*)$.

Algorithm 1: CFSSA-I

```
Input: A feasible instance of Problem (3)
List.create(POOL)
G_LB ← −∞; G_UB ← +∞
SearchDone ← FALSE
while SerachDone = FALSE do
    if G_UB − G_LB ≥ ε1 & (G_UB − G_LB)/G_UB ≥ ε2 then
        (x̄, ȳ) ← WSO(π, POOL, 0)
        if (x̄, ȳ) ≠ (null,null) then
            POOL.add((x̄, ȳ))
            if ∏_{i=1}^{p} ȳ_i^{π_i} > G_LB then
                (x*, y*) ← (x̄, ȳ)
                G_LB ← ∏_{i=1}^{p} ȳ_i^{π_i}
            G_UB ← ((∑_{i=1}^{p} π_i ȳ_i)/(∑_{i=1}^{p} π_i))^{∑_{i=1}^{p} π_i}
        else
            SearchDone ← TRUE
    else
        SearchDone ← TRUE
return (x*, y*)
```

In each iteration, CFSSA-I first checks whether $\mathrm{G_{UB}} - \mathrm{G_{LB}} \geq \varepsilon_1$ and $\frac{\mathrm{G_{UB}} - \mathrm{G_{LB}}}{\mathrm{G_{UB}}} \geq \varepsilon_2$ hold, where $\varepsilon_1, \varepsilon_2 \in (0, 1)$ are the user-defined absolute and relative optimality gap tolerances. If at least one of these conditions is not satisfied, then the algorithm terminates. Otherwise, the algorithm calls $\mathrm{WSO}(\boldsymbol{\pi}, \mathrm{POOL}, \mathbf{0})$ to compute $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$, i.e., a locally efficient solution and its image in the criterion space. If $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) = (null, null)$, i.e., the problem is infeasible, then the restricted feasible region is empty, and therefore, the algorithm terminates after returning $(\boldsymbol{x}^*, \boldsymbol{y}^*)$. However, if $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \neq (null, null)$ then $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ will be added to POOL to be used in the future iterations. Also, the global lower bound and upper bound will be updated based on Remark 2. Specifically, the algorithm first

updates the primal bound, meaning if $\prod_{i=1}^{p} \bar{y}_i^{\pi_i} > \mathrm{G_{LB}}$ then the best feasible solution known, i.e., $(\boldsymbol{x}^*, \boldsymbol{y}^*)$, will be set to $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ and G_{LB} will be set to $\prod_{i=1}^{p} \bar{y}_i^{\pi_i}$. Afterwards, the global dual bound will be updated by setting $\mathrm{G_{UB}}$ to $(\frac{\sum_{i=1}^{p} \pi_i \bar{y}_i}{\sum_{i=1}^{p} \pi_i})^{\sum_{i=1}^{p} \pi_i}$. A precise description of CFSSA-I can be found in Algorithm 1.

4.2. The Criterion Feasible Set Shrinking Algorithm II

We start this subsection by providing an observation about CFSSA-I, i.e., Algorithm 1. That is, in theory, it is possible to have an instance in which many feasible solutions in the decision space have exactly the same image in the criterion space. So, in CFSSA-I, it is possible that $\mathrm{WSO}(\boldsymbol{\pi}, \textsc{Pool}, \mathbf{0})$ returns the same feasible point, denoted by $\bar{\boldsymbol{y}}$, in the criterion space in different (consecutive) iterations. Of course, because we use no-good constraints, the solutions in the decision space are certainly different in each iteration of CFSSA-I but their image in the criterion space can be the same. This observation indicates that it is possible that CFSSA-I performs not well for such instances. As an aside, it is worth mentioning that in our computational study, this never happened. However, due to the importance of this issue, we next discuss a simple approach to resolve this issue when $\bar{\boldsymbol{y}} > \mathbf{0}$ and present a new algorithm, CFSSA-II, based on that. Note that we simply assume that $\bar{\boldsymbol{y}} \ngtr \mathbf{0}$ does not occur because based on our assumptions the optimal objective value of an IL-GMMP is strictly positive. Hence, one can simply add the condition $\boldsymbol{y} > \mathbf{0}$ to the formulation of an IL-GMMP.

We first note that at any iteration of Algorithm 1, after calling $\mathrm{WSO}(\boldsymbol{\pi}, \textsc{Pool}, \mathbf{0})$, $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ will be returned and will be added to Pool if $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \neq (null, null)$. In the remaining, we assume that $\bar{\boldsymbol{y}} > \mathbf{0}$. So, we know that at the next iteration of Algorithm 1, the following hypotenuse cut,

$$\sum_{i=1}^{p} \frac{\pi_i}{\bar{y}_i} y_i \geq \sum_{i=1}^{p} \pi_i,$$

exists in the model corresponding to $\mathrm{WSO}(\boldsymbol{\pi}, \textsc{Pool}, \mathbf{0})$. Observe that, as mentioned in Section 2, we could write this cut in the form of strict inequality based on Corollary 1. Hence, if we use the strict inequality, the feasible point $\bar{\boldsymbol{y}}$ can never be computed again in any iteration of Algorithm 1 and this resolves the issue that we mentioned above.

However, using such strict inequality gives rise to numerical issues. So, instead, we propose to simply check the remaining search region at the end of each iteration for identifying whether there exists any feasible point, denoted by $\hat{\boldsymbol{y}}$, which can strictly satisfy the hypotenuse cut associated with $\bar{\boldsymbol{y}}$. If there exists no such a feasible point, then the search is over. Otherwise, if $\hat{\boldsymbol{y}} > \mathbf{0}$, we can add a new hypotenuse cut based on $\hat{\boldsymbol{y}}$ and this results in avoiding the generation of $\bar{\boldsymbol{y}}$ in future iterations. Moreover, we can update the global primal bound based on $\hat{\boldsymbol{y}}$. Now the only remaining question is how to compute $\hat{\boldsymbol{y}}$ at the end of each iteration. In order to do so, we can call the

operation $\text{WSO}(\hat{\boldsymbol{\pi}}, \text{POOL}, \mathbf{0})$ where $\hat{\pi}_i = \dfrac{\pi_i}{\bar{y}_i}$ for all $i \in \{1, \cdots, p\}$. Obviously, the objective function of $\text{WSO}(\hat{\boldsymbol{\pi}}, \text{POOL}, \mathbf{0})$ is simply the left-hand-side of the hypotenuse cut associated with $\bar{\boldsymbol{y}}$. So, if the optimal objective value of $\text{WSO}(\hat{\boldsymbol{\pi}}, \text{POOL}, \mathbf{0})$ is not strictly greater than $\sum_{i=1}^{p} \pi_i$, which is the right-hand-side of the hypotenuse cut associated with $\bar{\boldsymbol{y}}$, then there is no feasible solution that can strictly satisfy the hypotenuse cut associated with $\bar{\boldsymbol{y}}$.

In light of the above, a precise description of CFSSA-II can be found in Algorithm 2 which is similar to Algorithm 1 but has some additional lines, i.e., see Lines 14-23.

Algorithm 2: CFSSA-II

```
1  Input: A feasible instance of Problem (3)
2  List.create(POOL)
3  G_LB ← −∞; G_UB ← +∞
4  SearchDone ← FALSE
5  while SerachDone = FALSE do
6      if G_UB − G_LB ≥ ε1 & (G_UB − G_LB)/G_UB ≥ ε2 then
7          (x̄, ȳ) ← WSO(π, POOL, 0)
8          if (x̄, ȳ) ≠(null,null) then
9              POOL.add((x̄, ȳ))
10             if ∏_{i=1}^{p} ȳ_i^{π_i} > G_LB then
11                 (x*, y*) ← (x̄, ȳ)
12                 G_LB ← ∏_{i=1}^{p} ȳ_i^{π_i}
13             G_UB ← ((∑_{i=1}^{p} π_i ȳ_i)/(∑_{i=1}^{p} π_i))^{∑_{i=1}^{p} π_i}
14             if ȳ > 0 then
15                 π̂ ← (π_1/ȳ_1, ..., π_p/ȳ_p)
16                 (x̂, ŷ) ← WSO(π̂, POOL, 0)
17                 if (x̂, ŷ) =(null, null) or ∑_{i=1}^{p} π̂_i ŷ_i ≤ ∑_{i=1}^{p} π_i then
18                     SearchDone ← TRUE
19                 else
20                     if ∏_{i=1}^{p} ŷ_i^{π_i} > G_LB then
21                         (x*, y*) ← (x̂, ŷ)
22                         G_LB ← ∏_{i=1}^{p} ȳ_i^{π_i}
23                     POOL.add((x̂, ŷ))
24         else
25             SearchDone ← TRUE
26     else
27         SearchDone ← TRUE
28 return (x*, y*)
```

4.3. The Criterion Feasible Set Shrinking Algorithm III

The third algorithm, the so-called CFSSA-III, is basically a significantly modified/enhanced version of the algorithm proposed by Saghand and Charkhgard (2019). The two main differences are as follows. First, the hypotenuse cut does not exist in the algorithm proposed by Saghand and Charkhgard (2019). Second, to compute a dual bound at each iteration, the algorithm proposed by Saghand and Charkhgard (2019) simply relaxes the integrality conditions and then solves the relaxed problem after reformulating it as a SOCP. So, not only additional SOCPs have to be solved in the algorithm proposed by Saghand and Charkhgard (2019) but also the obtained dual bounds are often significantly weaker. In addition, because of using SOCP solvers for computing

dual bounds, the algorithm proposed by Saghand and Charkhgard (2019) cannot directly handle non-unit geometric weights, i.e., because the size of the SOCP reformulation highly depends on the geometric weights.

Algorithm 3: CFSSA-III

Input: A feasible instance of Problem (3)
Queue.create(TREE)
List.create(POOL)
$\boldsymbol{l} \leftarrow \mathbf{0}$; $\text{UB} \leftarrow +\infty$
TREE.*add*$(\boldsymbol{l}, \text{UB})$
$\text{G}_{\text{LB}} \leftarrow -\infty$; $\text{G}_{\text{UB}} \leftarrow +\infty$
SearchDone $\leftarrow$ FALSE
while *not Queue.empty*(TREE) *&* *SerachDone* = FALSE **do**
  TREE.*PopOut*$(\boldsymbol{l}, \text{UB})$
  $\text{G}_{\text{UB}} \leftarrow \text{UB}$
  if $\text{G}_{\text{UB}} - \text{G}_{\text{LB}} \geq \varepsilon_1$ *&* $\frac{\text{G}_{\text{UB}} - \text{G}_{\text{LB}}}{\text{G}_{\text{UB}}} \geq \varepsilon_2$ **then**
    $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \leftarrow \text{WSO}(\boldsymbol{\pi}, \text{POOL}, \boldsymbol{l})$
    if $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \neq$*(null,null)* **then**
      $\text{UB} \leftarrow (\frac{\sum_{i=1}^{p} \pi_i \bar{y}_i}{\sum_{i=1}^{p} \pi_i})^{\sum_{i=1}^{p} \pi_i}$
      POOL.*add*$((\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}))$
      if $\prod_{i=1}^{p} \bar{y}_i^{\pi_i} > \text{G}_{\text{LB}}$ **then**
        $(\boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow (\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$
        $\text{G}_{\text{LB}} \leftarrow \prod_{i=1}^{p} \bar{y}_i^{\pi_i}$
      if $\text{UB} - \text{G}_{\text{LB}} \geq \varepsilon_1$ *&* $\frac{\text{UB} - \text{G}_{\text{LB}}}{\text{UB}} \geq \varepsilon_2$ **then**
        foreach $i \in \{1, 2, .., p\}$ **do**
          $\boldsymbol{l}^i \leftarrow \boldsymbol{l}$
          $l_i^i \leftarrow \bar{y}_i$
          TREE.*add*$(\boldsymbol{l}^i, \text{UB})$
  else
    SearchDone $\leftarrow$ TRUE
return $(\boldsymbol{x}^*, \boldsymbol{y}^*)$

CFSSA-III maintains a queue of nodes, denoted by TREE. Each node in the queue contains two vectors denoted by $\boldsymbol{l}$ and UB. The point $\boldsymbol{l}$ is a lower bound point in the criterion space that defines a search region, and UB is a dual bound for the associated search region. We initialize the queue (of nodes) by $\boldsymbol{l} = \mathbf{0}$ and $\text{UB} = +\infty$. Similar to CFSSA-I and CFSSA-II, CFSSA-III maintains three other pieces of information including POOL, G_{UB}, and G_{LB}. At the beginning, POOL is empty and we set $\text{G}_{\text{UB}} = +\infty$ and $\text{G}_{\text{LB}} = -\infty$. The algorithm explores the queue as long as it is nonempty and $\text{G}_{\text{UB}} - \text{G}_{\text{LB}} \geq \varepsilon_1$ and $\frac{\text{G}_{\text{UB}} - \text{G}_{\text{LB}}}{\text{G}_{\text{UB}}} \geq \varepsilon_2$. 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 $(\boldsymbol{l}, \text{UB})$. 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) because we have numerically observed that this strategy performs the best. Therefore, whenever a node is popped out, the G_{UB} can be updated as we know that the selected node contains the highest upper bound. The algorithm next calls $\text{WSO}(\boldsymbol{\pi}, \text{POOL}, \boldsymbol{l})$ to compute $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$. Afterwards, if $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \neq (null, null)$ then the algorithm does the following steps:

- It adds $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ to POOL.
- It sets UB to $(\frac{\sum_{i=1}^{p} \pi_i \bar{y}_i}{\sum_{i=1}^{p} \pi_i})^{\sum_{i=1}^{p} \pi_i}$ because this is a new dual bound for the node.
- The algorithm updates the global primal bound and the best solution found based on $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$.
- If there still exists chance of finding a better solution in this node, i.e., $\text{UB} - \text{G}_{\text{LB}} \geq \varepsilon_1$ and $\frac{\text{UB}-\text{G}_{\text{LB}}}{\text{UB}} \geq \varepsilon_2$, it will be decomposed into p new nodes. The difference between the new nodes are simply their corresponding search regions. Specifically, for each $i \in \{1, \ldots, p\}$, the node $(\boldsymbol{l}^i, \text{UB})$ will be added where $\boldsymbol{l}^i$ defines its associated search region such that $l^i_j = l_j$ for each $j \in \{1, \ldots, p\} \backslash \{i\}$ and $l^i_i = \bar{y}_i$.

In light of the above, a precise description of CFSSA-III can be found in Algorithm 3. As an aside, we note that a similar approach used for modifying CFSSA-I (to be transformed into CFSSA-II) can be applied to CFSSA-III. However, we do not report the performance of such a modified algorithm in this study because we numerically observed that it does not improve the solution time for our instances.

5. Computational study

In this section, we conduct an extensive computational study using three commercial solvers including CPLEX 12.7, Gurobi 8.1.1[1], and Xpress 8.5.6. Specifically, we implement CFSSA-I, CFSSA-II, and CFSSA-III in C++ and compare their performances when different commercial solvers are employed for solving single-objective integer linear programs arising during the course of these algorithms. Also, as shown in Section 2, an IL-GMMP can be reformulated as a mixed integer SOCP and therefore, can be solved by appropriate solvers. So, in this computational study, we also compare the performance of the mixed integer SOCP solvers of CPLEX, Gurobi, and Xpress. The instance generator and the C++ implementation of the algorithms used in this computational study can be found in `https://bit.ly/2KpVStw`. 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 3600 seconds is imposed for solving each instance for all algorithms.

In our computational study, a total of 2400 instances are generated and solved with different geometric weights using different algorithms/solvers. Hence, a total of 57600 experiments are conducted to complete this computational study. Specifically, for each $p \in \{2, 3, 4\}$, we generate 800

[1] We initially used Gurobi 8.1.0 during the course of this study. However, we observed that SOCP solver of Gurobi 8.1.0 is not reliable for our test instances. Specifically, for instances with $\sum_{i=1}^{p} \pi_i \notin \{2, 4\}$ or by deactivating its presolve process, it was immediately returning the optimal objective value of 1 (regardless of the instance). We reported this issue to Gurobi support forum. Fortunately, at the final stages of this research, Gurobi 8.1.1 was released that does not have the issue mentioned above. So, we rerun all experiments using Gurobi 8.1.1 in this paper.

instances and solve each one using different geometric weights shown in Table 1. For example, for instances with $p=2$, the vector of geometric weights, i.e., $\boldsymbol{\pi}$, can take 5 different values, i.e., $(1,1)$ $(1,2)$, $(2,1)$, $(1,3)$, and $(3,1)$, in this computational study.

Table 1 **Different geometric weights for each value of p**

	$\mathbf{p=4}$	$\mathbf{p=3}$	$\mathbf{p=2}$
$\sum_{i=1}^{p}\pi_i=4$	(1,1,1,1)	(2,1,1), (1,2,1), (1,1,2)	(1,3), (3,1)
$\sum_{i=1}^{p}\pi_i=3$	-	(1,1,1)	(1,2), (2,1)
$\sum_{i=1}^{p}\pi_i=2$	-	-	(1,1)

Now, we explain how the instances are generated. For each $p\in\{2,3,4\}$, two classes of instances including pure integer ($n_i=n$) and pure binary ($n_b=n$) instances are generated. Each class contains 20 subclasses of instances based on the dimensions of the matrix $A_{m\times n}$, and each subclass contains 20 instances. We assume that $m\in\{200,400,600,800,1000\}$ and $n=\alpha m$ where $\alpha\in\{0.5,1,1.5,2\}$. For example, our smallest subclass is 200×100 with $m=200$ constraints and $n=100$ variables, i.e., $\alpha=0.5$, and our largest subclass is 1000×2000 with $m=1000$ constraints and $n=2000$ variables, i.e., $\alpha=2$. 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 that are defining the set $\mathcal{X}$ are in the form of $\leq$ inequality 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 replaced by (a set of) binary decision variables when using our proposed algorithms, i.e., CFSSA-I, CFSSA-II, and CFSSA-III. However, this is not required when solving an instance using (mixed integer) SOCP solvers. Since the maximum value for components of vector $\boldsymbol{b}$ is 150 and the minimum value for entries of matrix A is 10, each integer decision variable can get a maximum value of 15. Therefore, the integer variable x_i is replaced by four binary derision variables and one additional constraint 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. Overall, for each of our pure integer instances, the number of decision variables increases by a factor of 4 when using our proposed algorithms.

In this computational study, we frequently use two types of charts for comparing the performance of different algorithms/solvers. The first one is a specific *boxplot* in which on the horizontal axis different algorithms are shown, and on the vertical axis the run time ratio is shown. Specifically, for constructing a boxplot, for each instance and for each algorithm, we need to compute the ratio of the run time of the algorithm on the instance to the minimum of the run times of all algorithms (used in the boxplot) on the instance. Hence, if the ratio is closer to one, then it is better. Another tool that we use is *performance profile* (Dolan and Moré 2002) which provides more details compared to a boxplot (but it does not look as nice/neat as a boxplot).

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 in this study. Similar to our boxplots, 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.

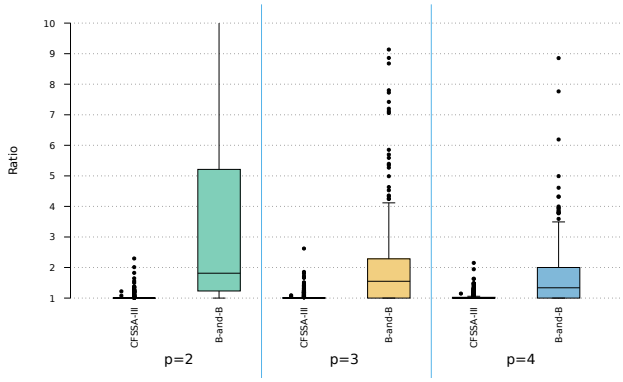


Figure 3 **Comparison between the algorithm proposed by Saghand and Charkhgard (2019), the so-called B-and-B, and CFSSA-III on pure binary instances for unit geometric weights**

As mentioned in Introduction and Section 4.3, CFSSA-III is a modified/enhanced version of the algorithm proposed by Saghand and Charkhgard (2019), the so-called *B-and-B*. Figure 3 is a boxplot for comparing the performance of these two algorithms on pure binary instances. Note that the C++ implementation of B-and-B is publicly available and uses CPLEX. Also, note that B-and-B cannot deal with the non-unit geometric weights directly, and that both algorithms replace any general integer decision variable with a new set of binary decision variables. Therefore, in order to

create the boxplot, we only used CPLEX in CFSSA-III, we set the geometric weights equal to one, and we only used the class of pure binary instances for each $p \in \{2, 3, 4\}$. Observe from Figure 3 that CFSSA-III significantly outperforms B-and-B. By comparing the medians, the run time of B-and-B is around 1.5 times greater than CFSSA-III. For some instances with $p = 2$, the run time of CFSSA-III is around 10 times better.

In light of the above, in the rest of this section, we de not report the results of B-and-B due to the reason that it does not perform better than CFSSA-III. Finally, it is worth mentioning that, in this computational study, the terms 'solve' and 'solve to optimality' have different meanings. The latter is used when an algorithm was able to find an optimal solution. The term 'solve' is used when an algorithm was able to find a feasible solution (other than zero) which may or may not be optimal. Note that by setting all decision variables equal to zero, a feasible solution with the objective value of zero will be constructed for our instances.

5.1. Two objectives ($\mathbf{p = 2}$)

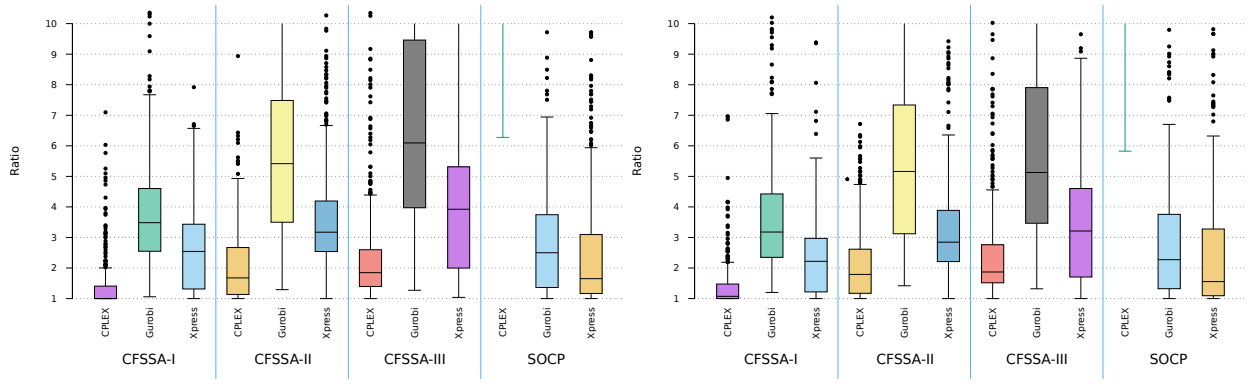
In this section, we compare the performance of the algorithms on instances with $p = 2$. CFSSA-I and CFSSA-II were able to solve all instances to optimality. However, CFSSA-III and the (mixed integer) SOCP solvers were not able to do so even for unit geometric weights. Table 2 shows a comparison between SOCP solvers and CFSSA-III for unit geometric weights. In this table, '#S' is the number of instances that an algorithm could solve, i.e., found a solution other than zero within the imposed time limit, '#Opt' is the number of instances solved to optimality, and '%Gap' shows the average (relative) optimality gap percentage of the instances that were not solved to optimality. Observe that both methods are able to find a feasible solution (other than zero) for all instances when different commercial solvers are embedded in them. However, SOCP solver of CPLEX is performing the worst. More than one third of the instances have the average optimality gap of around 60% when using SOCP solver of CPLEX.

Table 2 **Performance comparison for instances with $p = 2$ and $\boldsymbol{\pi} = (1, 1)$**

		CFSSA-III			**SOCP**		
		CPLEX	Gurobi	Xpress	CPLEX	Gurobi	Xpress
Binary	#S	400	400	400	400	400	400
	#Opt	395	392	394	249	400	400
	%Gap	0.06	0.09	0.06	58.92	0	0
Integer	#S	400	400	400	400	400	400
	#Opt	391	391	392	252	400	400
	%Gap	0.06	0.06	0.06	60.66	0	0

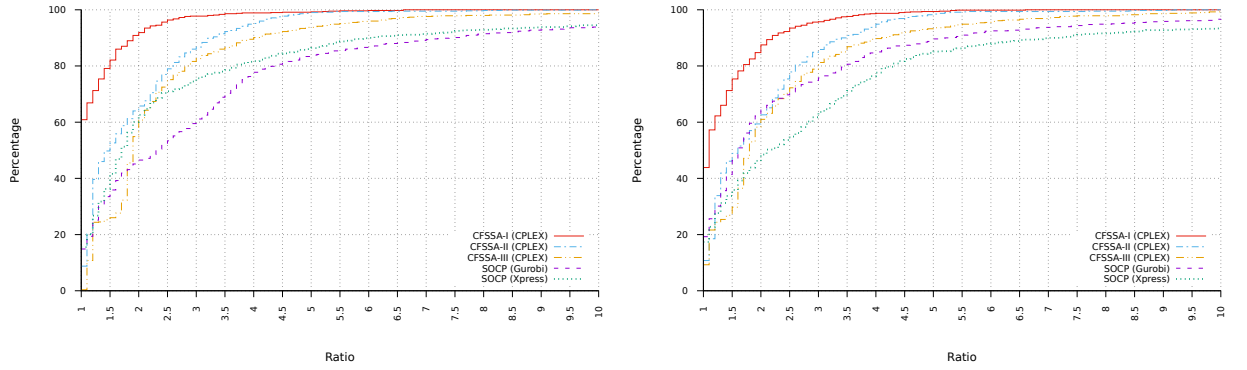
Figure 4 shows the run time boxplots of the algorithms on instances with $p = 2$ for unit geometric weights. From this figure, it is clear that all our proposed algorithms perform significantly better when using mixed integer linear programming solver of CPLEX. However, SOCP solver of the

CPLEX is the worst choice for solving a mixed integer SOCP transformation of an IL-GMMP. SOCP solver of Xpress seems to be the best choice for solving a mixed integer SOCP transformation of an IL-GMMP. However, by comparing the medians of the boxplots, we see that CFSSA-I when using CPLEX dominates SOCP solver of Xpress by a factor of around 1.5. The other two algorithms, CFSSA-II and CFSSA-III (when using CPLEX), are also competitive with SOCP solver of Xpress.



(a) Pure Binary (b) Pure Integer

Figure 4 **Performance comparison for instances with $p = 2$ and $\boldsymbol{\pi} = (1, 1)$**

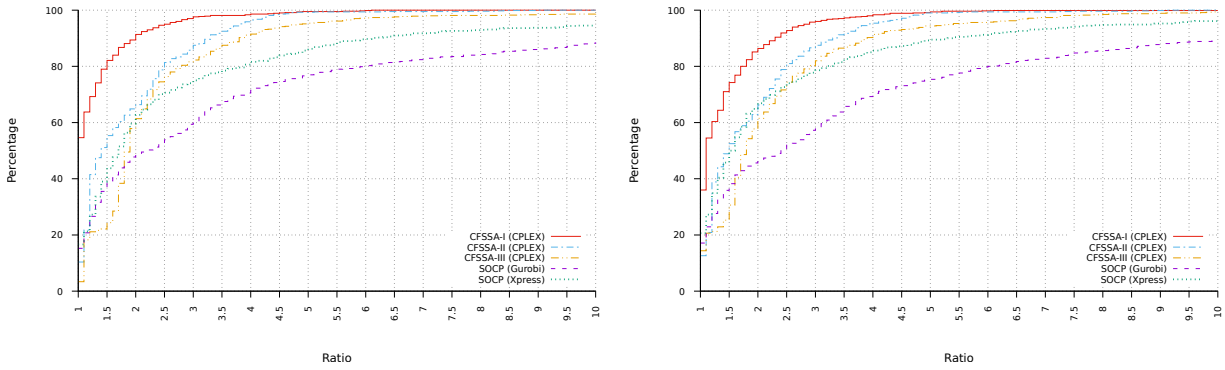


(a) Pure Binary (b) Pure Integer

Figure 5 **Performance profile for instances with $p = 2$ and $\boldsymbol{\pi} \in \{(2, 1), (1, 2)\}$**

Figures 5 and 6 show the run time performance profile of the best versions of the algorithms on the instances with $p = 2$ for non-unit geometric weights with $\sum_{i=1}^{p} \pi_i = 3$ and $\sum_{i=1}^{p} \pi_i = 4$, respectively. Note that for a few instances, Gurobi SOCP solver was better than Xpress SOCP solver. So, we have used both SOCP solvers of Xpress and Gurobi for creating these figures. Overall, CFSSA-I (when using CPLEX) performs significantly better than others. CFSSA-II and CFSSA-III are competitive (when using CPLEX) but CFSSA-II is slightly better. Overall, both CFSSA-II

and CFSSA-III can be considered as the second best algorithms. However, by comparing CFSSA-I and CFSSA-II, it is evident that around 20% of instances are solved at least 2.5 times faster by CSFSA-I in any of the figures. Moreover, the figures show that Xpress SOCP solver dominates the SOCP solver of Gurobi except for pure integer instances with $\sum_{i=1}^{p} \pi_i = 3$, i.e., Figure 5b. Overall, they are both either competitive with CFSSA-III or slightly worse than it.



(a) Pure Binary (b) Pure Integer

Figure 6 **Performance profile for instances with $p = 2$ and $\boldsymbol{\pi} \in \{(3,1),(1,3)\}$**

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

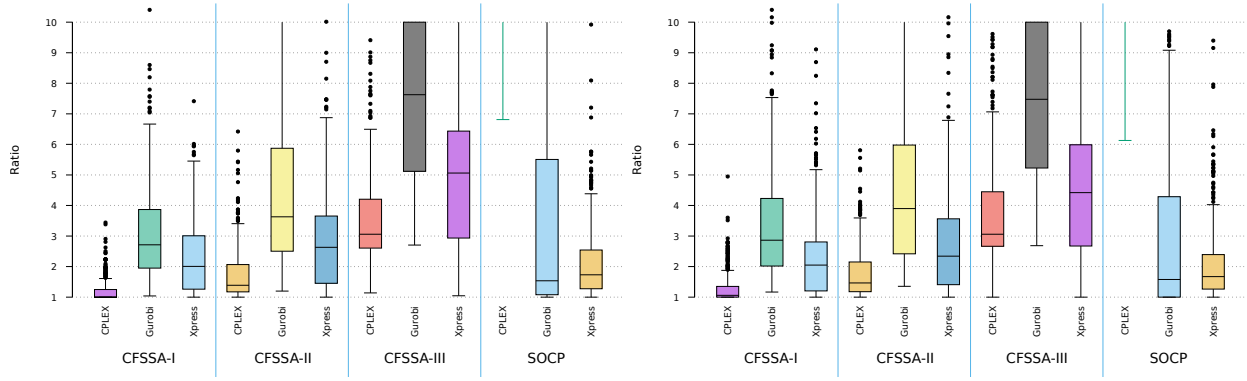
In this section, we compare the performance of the algorithms on instances with $p = 3$. Again, CFSSA-I and CFSSA-II were able to solve all instances to optimality within the imposed time limit. However, CFSSA-III and the (mixed integer) SOCP solvers were not able to do so even for unit geometric weights. Table 2 shows a comparison between SOCP solvers and CFSSA-III for unit geometric weights. We observe from the table that SOCP solver of Xpress was not able to solve only one of the instances to optimality for the class of pure integer instances. SOCP solver of Xpress could reach to the optimality gap of 24.85% for that particular instance. However, CFSSA-III was not able to solve 4 instances to optimality when employing CPLEX and had the average optimality gap of 0.34%.

Another interesting observation is that SOCP solver of CPLEX was not able to solve, i.e., find a feasible solution (other than zero), for 208 and 219 instances of the classes pure binary and pure integer, respectively. According to IBM ILOG CPLEX Optimization Studio (2016), the reason can be the convergence tolerance. Changing this tolerance to a smaller value may result in greater numerical precision of the solution, 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 default 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 the default value in this paper since it had the best performance for our instances.

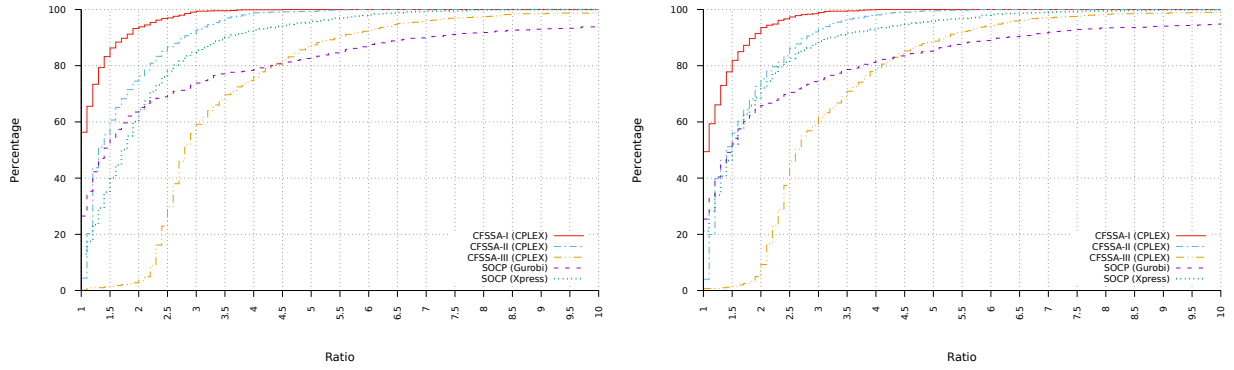
Table 3 **Performance comparison for instances with $p=3$ and $\boldsymbol{\pi}=(1,1,1)$**

		CFSSA-III			SOCP		
		CPLEX	Gurobi	Xpress	CPLEX	Gurobi	Xpress
Binary	#S	400	400	400	192	400	400
	#Opt	400	399	400	79	400	400
	%Gap	0	0.41	0	75.71	0	0
Integer	#S	400	400	400	181	400	400
	#Opt	396	394	397	71	400	399
	%Gap	0.34	0.37	0.33	74.39	0	24.84



(a) Pure Binary ($n=n_b$) (b) Pure Integer ($n=n_i$)

Figure 7 **Performance comparison for instances with $p=3$ and $\boldsymbol{\pi}=(1,1,1)$**



(a) Pure Binary ($n=n_b$) (b) Pure Integer ($n=n_i$)

Figure 8 **Performance profiles for instances with $p=3$ and $\boldsymbol{\pi}\in\{(2,1,1),(1,2,1),(1,1,2)\}$**

Figure 7 shows the run time boxplots of the algorithms on instances with $p=3$ for unit geometric weights. From the figure, it is clear that all our proposed algorithms perform significantly better

when using mixed integer linear programming solver of CPLEX. Again, SOCP solver of the CPLEX is the worst choice for solving a mixed integer SOCP transformation of an IL-GMMP. SOCP solver of Xpress seems to be the best choice for solving a mixed integer SOCP transformation of an IL-GMMP. However, by comparing the medians of the boxplots, we observe that CFSSA-I when using CPLEX dominates SOCP solver of Xpress by a factor of around 1.5. Finally, CFSSA-II (when using CPLEX) is competitive with SOCP solver of Xpress but CFSSA-III is not.

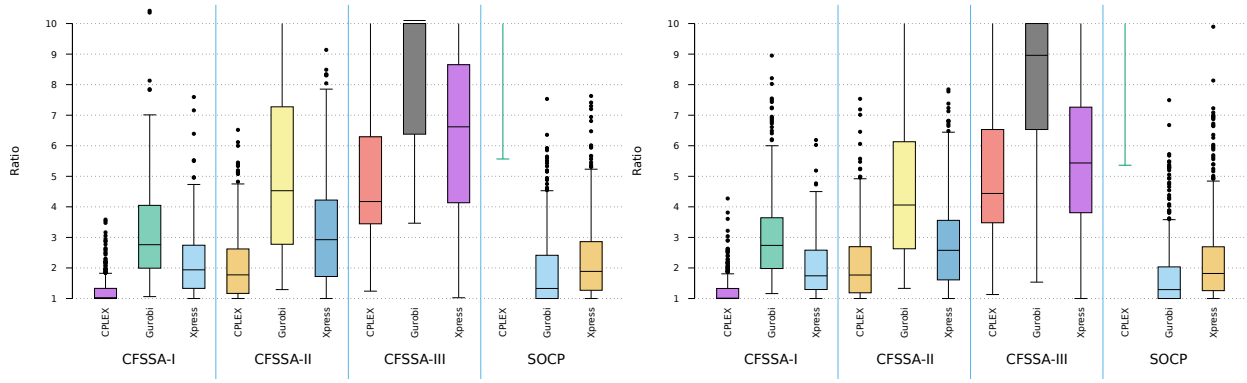
Figure 8 shows the run time performance profiles of the best versions of the algorithms on instances with $p = 3$ for non-unit geometric weights with $\sum_{i=1}^{p} \pi_i = 4$. Again, we have included both SOCP solvers of Gurobi and Xpress, because for some instances SOCP solver of Gurobi was performing better. From the figure, it is clear that CFSSA-I when using CPLEX is the best choice. For pure binary instances, the second best choice is CFSSA-II when using CPLEX. However, for pure integer instances, both CFSSA-II (when using CPLEX) and Xpress SOCP solver are the second best choice. Overall, we observe that the best choice has solved around 20% of instances at least 2 and 2.5 times faster than the second best choice in pure binary and pure integer classes, respectively. We also observe that unlike instances with $p = 2$ (see Figure 6), CFSSA-III is performing poorly compared to CFSSA-I and CFSSA-II. This is mainly because CFSSA-III uses a decomposition technique. So, as p increases, more child nodes can be created in each iteration.

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

In this section, we compare the performance of the algorithms on instances with $p = 4$. Similar to the results obtained for instances with $p \in \{2, 3\}$, CFSSA-I and CFSSA-II were able to solve all instances to optimality within the imposed time limit, i.e., one hour. However, CFSSA-III and the (mixed integer) SOCP solvers were not able to do so even for unit geometric weights. Table 2 shows a comparison between SOCP solvers and CFSSA-III for unit geometric weights. We observe from the table that SOCP solver of Gurobi was able to solve all instances to optimality. However, SOCP solver of Xpress was not able to solve 6 instances to optimality in total (three per class). The average optimality gap reported by those instances is around 43%. Also, CFSSA-III was not able to solve 11 instances to optimality in total when employing CPLEX. However, the average optimality gap reported by CFSSA-III (when using CPLEX) is around 4%. Finally, SOCP solver of CPLEX was not able to solve, i.e., find a feasible solution (other than zero), for 320 and 319 instances of the classes pure binary and pure integer, respectively.

Table 4 **Performance comparison for instances with $p = 4$ and $\boldsymbol{\pi} = (1, 1, 1, 1)$**

		CFSSA-III			**SOCP**		
		CPLEX	Gurobi	Xpress	CPLEX	Gurobi	Xpress
Binary	#S	400	400	400	80	400	400
	#Opt	395	392	396	19	400	397
	%Gap	4.38	3.91	5.26	85.74	0	42.09
Integer	#S	400	400	400	81	400	400
	#Opt	394	393	396	11	400	397
	%Gap	3.11	2.83	3.92	86.21	0	43.33



(a) Pure Binary ($n = n_b$) (b) Pure Integer ($n = n_i$)

Figure 9 **Performance comparison for instances with $p = 4$ and $\boldsymbol{\pi} = (1, 1, 1, 1)$**

Figure 9 shows the run time boxplots of the algorithms on instances with $p = 4$ for unit geometric weights. Again, we observe that all our proposed algorithms perform significantly better when using mixed integer linear programming solver of CPLEX. However, this time, although SOCP solver of Xpress performs well, it is not as good as the SOCP solver of the Gurobi for solving a mixed integer SOCP transformation of an IL-GMMP. By comparing the medians of the boxplots, we observe that CFSSA-I when using CPLEX dominates SOCP solver of Gurobi by a factor of around 1.3. Finally, it is clear that CFSSA-II (when using CPLEX) is as good as SOCP solver of Xpress.

5.4. Linearization

It is not hard to see that pure binary and pure integer instances can be linearized and solved directly by single-objective integer linear programming solvers. It is evident that linearizing pure binary instances with $p = 2$ and $\boldsymbol{\pi} = (1, 1)$ are the easiest cases. Consequently, the focus of this section will be only on such instances. As an example, consider the following objective function,

$$\max\ (x_1 + 3x_2)(x_2 + 2x_3) = x_1x_2 + 2x_1x_3 + 3x_2^2 + 6x_2x_3,$$

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_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,

$$x_ix_j := \left\{ q :\ q \le x_i,\ q \le x_j,\ x_i + x_j - 1 \le q \right\}$$

Note that, since the problem is in maximization form, the third constraint, i.e. $x_i + x_j - 1 \le q$, is redundant and can be eliminated. In light of this observation, the linearized objective function is as follows,

$$\max\ 3x_2 + q_1 + 2q_2 + 6q_3$$

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

We linearized all pure binary instances of Section 5.1 in the same way discussed above and solved them by CPLEX, Gurobi, and Xpress. Table 5 provides the detailed performance comparison on the linearized instances. Observe that the performance of the commercial solvers on linearized instances is very poor. In many instances, the solvers were unable to find even a feasible solution (other than 0) for linearized models but CFSSA-I (when using CPLEX) can solve even largest classes of instances to optimality in around 200 seconds on average. Interested readers may refer to Appendix A for details of the performance of CFSSA-I (when using CPLEX) and the best mixed integer SOCP solver on instances with unit geometric weights. Note that, pure binary instances with $p = 2$ and $\boldsymbol{\pi} = \mathbf{1}$ 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 and $\boldsymbol{\pi}$.

Table 5 Detailed performance of single-objective integer linear programming solvers on linearized binary instances with $\mathbf{p = 2}$ and unit geometric weights

$m \times n$	CPLEX				Gurobi				Xpress			
	#S	**#Opt**	**%Gap**	**T(sec.)**	**#S**	**#Opt**	**%Gap**	**T(sec.)**	**#S**	**#Opt**	**%Gap**	**T(sec.)**
200×100	20	20	-	10.11	20	20	-	6.92	20	20	-	67.09
200×200	20	20	-	316.87	20	20	-	216.88	20	20	-	1,303.96
200×300	20	19	64	1,546.60	20	19	56	710.93	20	-	94	3,600
200×400	20	1	92	3,566.87	20	13	81	2,661.54	20	-	98	3,600
Avg	20	15	39	1,360.11	20	18	34	899.07	20	10	48	2,142.76
400×200	20	20	-	122.28	20	20	-	29.93	20	20	-	867.07
400×400	20	12	88	2,920.83	20	20	-	344.62	20	-	99	3,600
400×600	20	-	97	3,600	20	20	-	1,730.64	20	-	99	3,600
400×800	4	-	100	3,600	20	-	97	3,600	20	-	99	3,600
Avg	16	8	71	2,560.78	20	15	24	1,426.30	20	5	74	2,916.77
600×300	20	20	-	405.62	20	20	-	91.87	20	-	97	3,600
600×600	20	1	97	3,555.49	20	20	-	1,220.49	20	-	99	3,600
600×900	4	-	100	3,600	20	-	95	3,600	20	-	99	3,600
600×1200	-	-	100	3,600	20	-	99	3,600	3	-	100	3,600
Avg	11	5.25	74	2,790.28	20	10	49	2,128.09	15.75	-	99	3,600
800×400	20	20	-	970.65	20	20	-	195.07	20	-	99	3,600
800×800	20	-	96	3,600	20	15	75	2,702.49	20	-	99	3,600
800×1200	-	-	100	3,600	20	-	99	3,600	4	-	100	3,600
800×1600	-	-	100	3,600	20	-	100	3,600	-	-	100	3,600
Avg	10	5	74	2,942.66	20	8.75	68	2,524.39	11	-	100	3,600
1000×500	20	19	77	1,571.99	20	20	-	354.70	20	-	99	3,600
1000×1000	13	-	98	3,600	20	8	82	3,287.97	20	-	100	3,600
1000×1500	-	-	100	3,600	20	-	99	3,600	-	-	100	3,600
1000×2000	-	-	100	3,600	20	-	100	3,600	-	-	100	3,600
Avg	8.25	4.75	94	3,093	20	7	70	2,710.67	10	-	100	3,600

6. Final remarks

We studied a subclass of GMMPs, the so-called IL-GMMPs, with a significant number of applications in many fields including but not limited to game theory, conservation planning, and system reliability. We developed three new multi-objective optimization based algorithms with two desirable characteristics: (1) they only solve single-objective integer linear programs and (2) they can directly deal with geometric weights. Using an extensive computational study, we demonstrated the efficacy of our proposed algorithms. We showed that the performance of our algorithms depends highly on the choice of a commercial solver employed for solving single-objective integer linear programs. Overall, for our test instances, CPLEX was shown to be the best choice for our proposed algorithms. We also illustrated that an IL-GMMP can be reformulated as a mixed integer SOCP but the size of such a reformulation depends on geometric weights. Overall, we showed that for our test instances, Xpress performs significantly better than CPLEX and Gurobi for solving such mixed integer SOCPs. However, even mixed integer SOCP solver of Xpress was shown to perform poorly compared to one of our algorithms, i.e., CFSSA-I. Finally, we showed that although one can linearize the objective function of an IL-GMMP (by introducing new sets of variables and constraints), commercial solvers struggle to solve even small IL-GMMPs using the linearized formulation. Next, two future research directions are explained.

The first future research direction of this study is about how to customize the proposed algorithms for solving MIL-GMMPs. One simple idea would be to employ the approach proposed by Saghand and Charkhgard (2019). In this approach, an extra primal-bound updating operation should be added to each algorithm. This operation should be called after computing a feasible solution $(\boldsymbol{x}', \boldsymbol{y}')$ by $\text{WSO}(\boldsymbol{w}, \text{POOL}, \boldsymbol{l})$ at any iteration. In this operation, we set the values of only integer decision variables of a MIL-GMMPs to those in $\boldsymbol{x}'$. By doing so, a reduced continuous problem will be generated which is basically a L-GMMP. So, we can reformulate such a L-GMMP as a SOCP and solve it accordingly to obtain the best feasible solution associated with the integer support vector of $\boldsymbol{x}'$. Therefore, that feasible solution should be used for updating the global primal bound (if it is better).

Although the approach mentioned above is theoretically correct, it is certainly worth investigating its performance in practice. In fact it is possible that solving such SOCPs creates numerical issues for large values of p and/or the geometric weights. This is because an optimal objective value of a L-GMMP (or in general any GMMP) can easily become a very small/large number as p and/or the geometric weights increase. So, to obtain more precise solutions, the optimality gap should be set to very small values and this by itself can be a potential source of numerical issues. Hence, any solution approach that solves optimization problems (such as the SOCP reformulation) in which the objective function of a GMMP, i.e., the multiplicative function, exists in them is not

probably reliable/precise for large values of p and/or the geometric weights. Note that none of the optimization problems solved in our proposed algorithms (in this study) deals with multiplicative function. So, in some sense, our proposed approaches are expected to be less sensitive to large values of p and/or the geometric weights. So, customizing these algorithms for solving MIL-GMMPs is better to be done in a way that this property to be preserved.

The second future research direction of this study is about how to explore the idea of developing multi-objective optimization based algorithms for other classes of single-objective optimization problems. In particular, GMMPs with unit geometric weights appear to be closely related to *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 GMMP with the unit geometric weights from *max* to *min* a minimum multiplicative program is obtained. Note that L-GMMPs can be solved in polynomial time. However, it is known that a minimum multiplicative program is NP-hard even when all constraints are linear and all decision variables are continuous (Shao and Ehrgott 2016). Therefore, because of this significant difference, we did not consider this class of optimization problems in this study. However, it is worth studying whether similar approaches can be developed for minimum multiplicative programs involving only linear constraints and some integer decision variables.

Appendix A: A detailed comparison between the best mixed integer SOCP solver and the best version of CFSSA-I on instances with unit geometric weights

In Tables 6 and 7, '#IP' is the number of single-objective integer linear programs solved by CFSSA-I, and 'T(sec.)' is the solution time in seconds.

Table 6 Pure binary instances with unit geometric weights

$m \times n$	$p=2$			$p=3$			$p=4$		
	CFSSA-I (CPLEX)		**SOCP** (Xpress)	**CFSSA-I** (CPLEX)		**SOCP** (Xpress)	**CFSSA-I** (CPLEX)		**SOCP** (Gurobi)
	#IP	**T(sec.)**	**T(sec.)**	**#IP**	**T(sec.)**	**T(sec.)**	**#IP**	**T(sec.)**	**T(sec.)**
200×100	2.30	0.50	0.45	2.85	0.67	0.62	3.20	0.81	1.14
200×200	2.35	0.98	1.05	2.60	1.08	1.58	3.05	1.72	2.85
200×300	2.30	1.52	1.90	2.80	3.39	3.90	2.80	3.15	4.49
200×400	2.05	3.39	3.90	2.80	8.06	7.74	3.00	6.60	7.93
Avg	2.25	1.60	1.82	2.76	3.30	3.46	3.01	3.07	4.10
400×200	2.45	2.16	2.21	2.70	2.55	3.11	3.05	3.92	9.57
400×400	2.35	4.84	7.35	2.45	7.78	10.61	2.85	10.21	16.12
400×600	2.10	12.27	15.80	2.55	17.80	27.66	2.90	28.39	23.92
400×800	2.20	21.39	23.49	2.35	31.42	46.32	3.05	52.02	36.54
Avg	2.28	10.17	12.22	2.51	14.89	21.92	2.96	23.63	21.54
600×300	2.35	5.37	6.21	2.35	6.05	8.73	2.80	11.77	20.45
600×600	2.25	36.36	32.98	2.05	35.57	50.19	2.90	65.19	45.91
600×900	2.10	61.35	77.90	2.60	89.05	120.77	2.60	97.18	88.98
600×1200	2.05	91.70	168.43	2.70	140.40	288.75	2.30	122.53	131.35
Avg	2.19	48.69	71.38	2.43	67.77	117.11	2.65	74.17	71.68
800×400	2.30	8.43	11.41	2.10	11.39	17.50	3.20	22.65	44.24
800×800	2.00	61.32	60.54	2.20	83.12	99.29	3.00	126.62	87.56
800×1200	2.25	140.89	195.04	2.15	150.30	325.68	2.55	179.75	198.21
800×1600	2.85	300.91	464.76	2.60	345.53	838.86	2.40	379.69	319.68
Avg	2.35	127.89	182.94	2.26	147.58	320.33	2.79	177.18	162.42
1000×500	2.40	21.19	20.88	2.30	16.40	29.34	2.70	31.11	71.47
1000×1000	2.60	122.98	102.95	2.85	192.68	195.04	3.50	248.37	176.51
1000×1500	3.55	339.47	420.36	2.35	284.43	595.67	2.50	290.94	330.39
1000×2000	2.05	332.24	1,079.01	2.40	554.23	1,623.30	2.25	479.06	495.21
Avg	2.65	203.97	405.80	2.48	261.94	610.84	2.74	262.37	268.40

Table 7 Pure integer instances with unit geometric weights

$m \times n$	$p=2$			$p=3$			$p=4$		
	CFSSA-I (CPLEX)		**SOCP** (Xpress)	**CFSSA-I** (CPLEX)		**SOCP** (Xpress)	**CFSSA-I** (CPLEX)		**SOCP** (Gurobi)
	#IP	**T(sec.)**	**T(sec.)**	**#IP**	**T(sec.)**	**T(sec.)**	**#IP**	**T(sec.)**	**T(sec.)**
200×100	2.15	0.56	0.46	3.00	0.96	0.68	3.60	1.26	1.35
200×200	2.50	1.38	1.21	2.70	1.73	1.79	2.65	2.33	2.59
200×300	2.20	1.79	1.95	2.80	3.67	4.00	3.30	4.77	5.77
200×400	2.35	5.45	4.29	2.40	7.01	7.68	3.05	11.01	8.58
Avg	2.30	2.30	1.98	2.73	3.34	3.54	3.15	4.84	4.57
400×200	2.35	2.40	2.42	3.10	4.13	3.50	2.75	3.86	8.39
400×400	2.20	7.17	7.59	2.50	10.44	13.33	3.30	13.64	16.21
400×600	2.05	8.94	12.56	2.70	21.44	22.50	2.60	26.61	24.59
400×800	2.05	21.76	21.36	2.55	29.70	34.74	3.10	54.32	41.66
Avg	2.16	10.07	10.98	2.71	16.42	18.52	2.94	24.61	22.71
600×300	2.20	5.50	6.28	2.35	5.96	9.65	3.40	12.09	18.23
600×600	1.95	28.46	32.78	2.20	38.87	52.66	2.80	54.01	48.92
600×900	1.95	57.68	85.95	2.20	74.87	122.45	2.60	89.27	89.63
600×1200	2.00	98.96	166.15	2.75	148.99	286.57	3.15	177.73	151.61
Avg	2.03	47.65	72.79	2.38	67.17	117.83	2.99	83.28	77.10
800×400	2.20	11.78	13.42	2.50	17.46	20.42	2.70	19.53	38.47
800×800	1.90	58.20	76.84	2.70	103.51	129.93	2.85	120.76	95.30
800×1200	3.20	220.34	289.09	2.65	169.54	338.19	2.20	172.06	202.87
800×1600	2.75	338.12	768.56	2.85	353.13	968.90	2.35	284.82	314.93
Avg	2.51	157.11	286.98	2.68	160.91	364.36	2.53	149.29	162.90
1000×500	2.55	17.68	20.62	2.85	28.64	29.14	3.00	36.74	71.29
1000×1000	2.05	118.30	152.73	2.35	156.05	235.28	2.70	175.11	181.26
1000×1500	1.80	160.09	409.84	2.70	330.57	696.16	2.90	368.95	368.98
1000×2000	4.25	804.30	1,130.13	2.80	584.50	1,905.47	2.40	522.43	525.06
Avg	2.66	275.09	428.33	2.68	274.94	716.51	2.75	275.81	286.65

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 (2017) 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.

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.

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.

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

Engau A, Wiecek MM (2008) Interactive coordination of objective decompositions in multiobjective programming. *Management Science* 54(7):1350–1363.

Erlebach T, Kellerer H, Pferschy U (2002) Approximating multiobjective knapsack problems. *Management Science* 48(12):1603–1612.

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).

Johnson MP, Hurter AP (2000) Decision support for a housing mobility program using a multiobjective optimization model. *Management Science* 46(12):1569–1584.

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

Kalai E (1977) Nonsymmetric Nash solutions and replications of 2-person bargaining. *International Journal of Game Theory* 6(3):129–133.

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.

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.

Phelps SP, Köksalan M (2003) An interactive evolutionary metaheuristic for multiobjective combinatorial optimization. *Management Science* 49(12):1726–1738.

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

Saghand PG, Charkhgard H (2019) A criterion space search algorithm for mixed integer linear maximum multiplicative programs: A multi-objective optimization approach. *Preprint* `http://www.optimization-online.org/DB_FILE/2019/01/7031.pdf`.

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, Kouvelis P (2005) The multiobjective discrete optimization problem: A weighted min-max two-stage optimization approach and a bicriteria algorithm. *Management Science* 51(10):1572–1581.

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.

Steele JM (2004) *The Cauchy-Schwarz Master Class: An Introduction to the Art of Mathematical Inequalities* (Cambridge University Press), URL `http://dx.doi.org/10.1017/CBO9780511817106`.

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

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.

Wallenius J, Dyer JS, Fishburn PC, Steuer RE, Zionts S, Deb K (2008) Multiple criteria decision making, multiattribute utility theory: Recent accomplishments and what lies ahead. *Management Science* 54(7):1336–1349.

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