June 2021

Maximum Multiplicative Programming: Theory, Algorithms, and Applications

Payman Ghasemi Saghand
University of South Florida

Maximum Multiplicative Programming: Theory, Algorithms, and Applications

by

Payman Ghasemi Saghand

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Industrial Engineering
Department of Industrial and Management Systems Engineering
College of Engineering
University of South Florida

Date of Approval:
June 20, 2021

Keywords: Geometric Mean Optimization, Optimization Over the Efficient Set,
Multi-objective Optimization, Nash Bargaining Solution, Nash Social Welfare

Dedication

To my wonderful wife and my beloved family for their unconditional love and support.

Acknowledgments

I would like to start by extending my sincere appreciation to my advisor, Dr. Hadi Charkhgard. He was the main reason that I chose the University of South Florida (USF) for my Ph.D., and his permanent support, guidance, and motivation made my Ph.D. the most fruitful portion of my academic journey. It has been indeed a pleasure to work and study under his supervision, and to work with the members of his research team, the Multi-objective Optimization Laboratory.

I would also like to thank all the brilliant people that I had the opportunity to work with, specially Dr. Julien Martin, Dr. Changhyun Kwon, Dr. Susana Lai-Yuen, and Dr. Fabian Rigterink. I also want to acknowledge the financial support by the National Science Foundation under Grant No 1849627, and by the United States Geological Survey - Southeast Ecological Science Center under the Award No 140G0119P0279.

I wish to express my deepest gratitude to my family, to whom I dedicated this dissertation. My parents, Heidar and Ozra, for all their sacrifice and unconditional love. I hope that I can be a good son for you and make you proud. My parents-in-law, Mohammad Reza and Parvin, for their heart-warming support and kindness. My sister, Esmat, whom I am very lucky to have in my life. And finally, my gorgeous wife, Forouzandeh, who brought light to my life with her love, support, and patience and made our life the happiest.

Table of Contents

List of Tables

List of Figures

Abstract

This dissertation presents three different contributions to an important class of optimization problems known as Multiplicative Programs (MPs). The first group of contributions contains the development and analysis of several multi-objective optimization-based based algorithms designed to find the optimal solution of Mixed Integer Linear MPs. As for the second group, the application of a special class of MPs in radiotherapy planning is presented. Finally, in the last group, a new technique for conducting the multiplication process in the objectives of MPs is presented. Using this technique, we introduce a family of novel solution methods that are capable of solving both maximum and minimum MPs.

Regarding the first group of contributions, we show that MPs can be viewed as special cases of the problem of optimization over the efficient set in Multi-objective optimization. Based on this observation, we develop several solution approaches with their own unique properties. In the first solution approach, we embed an existing algorithm capable of solving continuous MPs with two multiplying terms in a branch-and-bound framework to solve mixed integer instances. In our second approach, we develop a criterion space search algorithm for solving any mixed integer MPs with any number of multiplying terms. In our last solution approach, we develop three different algorithms that, in addition to solving any mixed integer MPs with any number of multiplying terms, are capable of handling the multiplying terms with different powers. Specifically, the advantage of our last solution approach is that it handles the powers with the minimal impact on the computational complexity in practice.

In the second group of contributions, we study the fluency map optimization problem in Intensity Modulated Radiation Therapy (IMRT) from a cooperative game theory point of view. We consider the cancerous and healthy organs in a patient's body as players of a game, where cancerous organs seek to eliminate the cancerous cells and healthy organs seek to receive no harm. We balance these trade-off by transforming the fluency map optimization problem into a bargaining game. Then, using the concept of Nash Social Welfare, we find a solution for our bargaining game by creating and optimizing an MP. The importance of our solution is that it simultaneously satisfies efficiency and fairness in the trade-offs. Finally, we demonstrate the advantages of our proposed approach by implementing it on two different cancer cases.

In the third and last group of contributions, we present the novel idea of binary-encoding the multiplication operation analogously to how a computer conducts it internally. Based on this idea, we develop a new family of solution methods for MPs. One of our methods is to solve the multiplicative programs bit-by-bit, i.e., iteratively computing the optimal value of each bit of the objective function. In an extensive computational study, we explore a number of solution methods that solve MPs faster and more accurately.

Chapter 1: Introduction

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

$$\max\left\{\prod_{i=1}^{p} y_i^{\pi_i}(\boldsymbol{x}) : \ \boldsymbol{x} \in \mathcal{X}, \ \boldsymbol{y}(\boldsymbol{x}) \geqslant \mathbf{0}\right\}, \tag{1.1}$$

where $\mathcal{X} \subseteq \mathbb{R}^n$ represents the set of feasible solutions which 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 vector of so-called *geometric weights* where $\boldsymbol{y}(\boldsymbol{x}) \geqslant \mathbf{0}$ and $\boldsymbol{\pi} > \mathbf{0}$, i.e., $y_i(\boldsymbol{x}) \geqslant 0$ and $\pi_i > 0$ for all $i \in \{1, \ldots, p\}$. The objective function of a GMMP is sometimes referred to as the Nash social welfare function in the literature [2]. 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 finite 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. We will also explain how our proposed algorithms can be easily modified to solve any MIL-GMMP by customizing the technique developed by [1]. The reason that we do not focus on MIL-GMMPs directly in this study is that when using the 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. Such non-linear convex optimization problems have some undesirable computational characteristics, for instance they cannot directly handle fractional geometric weights.

The proposed exact algorithms in this study are developed based on a critical observation that an optimal solution of Problem (1.1) 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,

$$\max\left\{y_1(\boldsymbol{x}),\dots,y_p(\boldsymbol{x}):\ \boldsymbol{x}\in\mathcal{X},\ \boldsymbol{y}(\boldsymbol{x})\geqslant 0\right\}. \tag{1.2}$$

Specifically, by maximizing the function $\prod_{i=1}^{p} y_i^{\pi_i}(\boldsymbol{x})$ over the set of efficient solutions of Problem (1.2), an optimal solution of Problem (1.1) can be obtained. We will prove this observation in this thesis but similar proofs can be found in [3, 4, 1], and [5, 6]. Overall, this critical observation indicates that Problem (1.1) 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) [7, 8, 9, 10].

1.1 Motivation

The main motivation of this study is based on the 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.

1.1.1 Game Theory

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 [11]. 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 [5, 6]. 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. The geometric weights in the context of bargaining problems/games indicate the negotiation power of the players. Also, integer decision variables arise in the context of discrete bargaining games, e.g. fair allocation of indivisible goods (for example artworks and jewelry) among multiple players/agents[1.1] [12, 13, 14]. 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 [3, 15, 16, 17], and [18] for further details about other applications.

1.1.2 Multi-objective Optimization

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 [19, 20, 21, 22, 23, 24, 25, 26]. For example, in business settings, there may be trade-offs between long-term profits and short-term cash

[1.1] http://www.spliddit.org/apps/goods 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 [7, 8]. An approach that can resolve this issue is finding a preferred solution among the set of efficient solutions directly [9, 7]. This approach is known as optimization over the efficient set, which is a global optimization problem [10]. So, in order to directly compute a Pareto-optimal solution that can balance the conflicting objectives, a multi-objective optimization problem can be transformed into a GMMP [27, 28].

1.1.3 Conservation Planning

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 [29]. The feasible set of such a problem can be easily formulated using binary decision variables and some constraints [30, 31]. 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 $\boldsymbol{x} \in \mathcal{X}$ [32, 33, 34]. The geometric weights represent the importance level of species from the view point of decision maker(s).

1.1.4 Other Applications

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 [35, 36, 37, 3].

For example, when using maximum likelihood estimation for nested logit models, binary decision variables arise [38, 39]. However, geometric weights are not applicable to maximum likelihood estimation to the best of our knowledge. Similarly, 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 $\boldsymbol{x} \in \mathcal{X}$. For system reliability, binary decision variables are required for indicating whether a particular component should be selected for a particular subsystem or not. Also, the geometric weights represent the importance level of each subsystem. Interested readers may refer to Appendix C2, to see three specific case studies from three different fields of study, i.e. game theory, health care, and transportation that give rise to IL-GMMPs.

1.2 Contributions of the Thesis

Throughout the body of this thesis, we developed new solution approaches for L-GMMP that could solve the problems faster and with higher precision. In addition, we applied the concept of L-GMMP to a new field, i.e. radiotherapy planning. Our research resulted in 5 major contributions: the publication of 3 journal articles and the submission of 2 journal articles (at the time of writing this thesis). Our contributions are listed below:

- P1: Ghasemi Saghand, P., & 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.* Volume 101, Pages 263-274, https://doi.org/10.1016/j.cor.2018.08.004.
- P2: Ghasemi Saghand, P., & Charkhgard, H.. (2021). A criterion space search algorithm for mixed integer linear maximum multiplicative programs: a multiobjective optimization approach. *International Transactions in Operational Research*, https://doi.org/10.1111/itor.12964.

- P3: Ghasemi Saghand, P., & Charkhgard, H.. (2021). Exact Solution Approaches for Integer Linear Generalized Maximum Multiplicative Programs Through the Lens of Multi-objective Optimization, (Submitted and under review).

- P4: Ghasemi Saghand, P., & Charkhgard, H.. (2021). A cooperative game solution approach for intensity modulated radiation therapy design: Nash Social Welfare optimization. *Physics in Medicine & Biology.* Volume 66, Number 7, https://doi.org/10.1088/1361-6560/abed95.

- P5: Ghasemi Saghand, P., & Rigternik, F., & Charkhgard, H.. (2021). Solving Multiplicative Programs by Binary-encoding the Multiplication Operation, (Submitted and under review).

1.3 Outline of the Thesis

The remaining content of this thesis is organized as follows:

- In Chapter 2, we present our paper P1. In this chapter, we focused on solving IL-MMPs with only 2 multiplying terms by embedding the algorithm developed in [3] in an effective branch-and-bound framework. We also develop several enhancement techniques including preprocessing, cuts, branching strategies, and node selection strategies to further improve the efficiency of our branch-and-bound. Through our computational study, we demonstrate that the proposed branch-and-bound algorithm outperforms a commercial mixed integer SOCP solver.

- In Chapter 3, we present our paper P2. In this chapter, we developed a new solution approach capable of solving IL-MMP with any number of multiplying terms. Our proposed algorithm is a criterion space search algorithm as it finds the optimal solution by considering the MIL-MMP as a multi-objective optimization problem and working on the space of objective functions. Finally, by means of our computational study, we depict that our proposed algorithm significantly outperforms CPLEX SOCP solver.

- In Chapter 4, we present our paper P3. In this chapter, 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 finite number of single-objective integer linear programs to compute an optimal solution of an IL-GMMP (which is nonlinear). 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.

- In Chapter 5, we present our paper P4 where we study the fluency map optimization problem in Intensity Modulated Radiation Therapy (IMRT) from a cooperative game theory point of view. We consider the cancerous and healthy organs in a patient's body as players of a game, where cancerous organs seek to eliminate the cancerous cells and healthy organs seek to receive no harm. We balance these trade-off by transforming the fluency map optimization problem into a bargaining game. Then, using the concept of Nash Social Welfare (NSW), we find a solution for our bargaining game that satisfies efficiency and fairness. Finally, we demonstrate the advantages of our our proposed approach by implementing it on two different cancer cases.

- In Chapter 6, we present our paper P5. In this chapter, we try to solve maximum and minimum multiplicative programs from a new approach. The main idea is to binary-encode the multiplication operation analogously to how a computer conducts it internally. Based on this idea, we develop a new family of solution methods for multiplicative programs. One of our methods is to solve the multiplicative programs bit-by-bit, i.e., iteratively computing the optimal value of each bit of the objective function. In an extensive computational study, we explore a number of solution methods that solve multiplicative programs faster and more accurately.

- Finally, in Chapter 7, we present the summary of the chapters discussed in this thesis, provide their concluding remarks, and establish some guidelines for the future research.

Chapter 2: A Branch-and-Bound Algorithm for a Class of Mixed Integer Linear Maximum Multiplicative Programs: A Bi-objective Optimization Approach

The copyright permissions for reuse previously published material in this chapter can be found in Appendix A1.

The main contribution of this chapter is extending the algorithm proposed by [3] to solve any mixed integer linear maximum multiplicative program with $p = 2$, i.e., those for which some of the decision variables have to take integer values. We propose an effective branch-and-bound algorithm which employs the power the algorithm developed by [3] for solving mixed integer instances. We also develop several preprocessing and cut generating techniques to improve the solution time of the proposed algorithm. Moreover, the effects of several branching and node selecting strategies are explored. A computational study shows that, for large-sized mixed binary instances, our proposed approach outperforms a commercial solver, i.e., CPLEX Mixed Integer SOCP solver, by a factor of around two on average. Also, for general mixed integer instances, our proposed algorithm outperforms CPLEX Mixed Integer SOCP solver not only on large-sized instances but also on small-sized instances by a factor of two on average.

The structure of this chapter is organized as follows. In Section 2.1, we provide some preliminaries and explain a high-level description of the algorithm proposed by [3]. In Section 2.2, we explain our proposed branch-and-bound algorithm in detail. In Section 2.3, some branching strategies are introduced. In Section 2.4, some node selecting strategies are presented. In Section 2.5, we explain some potential enhancement techniques. In Section 2.6, we conduct an extensive computational study. The concluding remarks of this chapter are provided in Section 7.1.

2.1 Preliminaries

A Mixed Integer Linear Maximum Multiplicative Program with $p = 2$ can be stated as follows:

$$\begin{aligned} \max\ & \prod_{i=1}^{2} y_i \\ \text{s.t. } & \boldsymbol{y} = D\boldsymbol{x} + \boldsymbol{d} \\ & A\boldsymbol{x} \leqslant \boldsymbol{b} \\ & \boldsymbol{x}, \boldsymbol{y} \geqslant \boldsymbol{0}, \quad \boldsymbol{x} \in \mathbb{R}^{n_c} \times \mathbb{B}^{n_b} \times \mathbb{Z}^{n_i}, \quad \boldsymbol{y} \in \mathbb{R}^2, \end{aligned} \tag{2.1}$$

where n_c, n_b, and n_i represent the number of continuous, binary, and integer decision variables, respectively. Also, D is a $2 \times n$ matrix where $n := n_c + n_b + n_i$, $\boldsymbol{d}$ is a 2-vector, A is an $m \times n$ matrix, and $\boldsymbol{b}$ is an m-vector.

The focus of this study is on solving Problem (2.1). We refer to the set $\mathcal{X} := \{\boldsymbol{x} \in \mathbb{R}^n : A\boldsymbol{x} \leqslant \boldsymbol{b},\ \boldsymbol{x} \geqslant \boldsymbol{0}\}$ as *the feasible set in the decision space* and to the set $\mathcal{Y} := \{\boldsymbol{y} \in \mathbb{R}^2 : \boldsymbol{x} \in \mathcal{X},\ \boldsymbol{y} = D\boldsymbol{x} + \boldsymbol{d},\ \boldsymbol{y} \geqslant \boldsymbol{0}\}$ as *the feasible set in the criterion space*. We assume that $\mathcal{X}$ is bounded (which implies that $\mathcal{Y}$ is compact) and the optimal objective value of the problem is strictly positive, i.e., 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).

Definition 2.1. A feasible solution $\boldsymbol{x} \in \mathcal{X}$ is called *efficient*, if there is no other $\boldsymbol{x}' \in \mathcal{X}$ such that $y_1 \leqslant y'_1$ and $y_2 < y'_2$ or $y_1 < y'_1$ and $y_2 \leqslant y'_2$ 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 2.1. *An optimal solution of Problem* (2.1)*, denoted by* $\boldsymbol{x}^*$*, is an efficient solution and therefore its corresponding image in the criterion space, denoted by* $\boldsymbol{y}^*$ *where* $\boldsymbol{y}^* := D\boldsymbol{x}^* + \boldsymbol{d}$*, is a nondominated point.*

Proof. Suppose that $\boldsymbol{x}^*$ is an optimal solution of Problem (2.1) 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 either $y_1^* \leqslant y_1$ and $y_2^* < y_2$ or $y_1^* < y_1$ and $y_2^* \leqslant y_2$ (where $\boldsymbol{y} := D\boldsymbol{x} + \boldsymbol{d}$). Also, by assumptions of Problem (2.1) we know that $\boldsymbol{y}^* > \boldsymbol{0}$. Therefore, we must have that $0 < y_1^* y_2^* < y_1 y_2$. Consequently, $\boldsymbol{x}^*$ cannot be an optimal solution (a contradiction). □

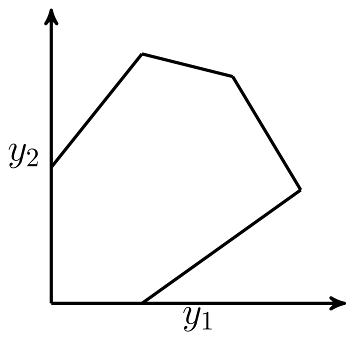
Proposition 2.1 implies that Problem (2.1) is equivalent to $\max_{\boldsymbol{y} \in \mathcal{Y}_N} y_1 y_2$ and this is precisely optimization over the efficient set. [3] used this observation and developed an algorithm, which we refer to as CST (which comes from the names of its authors Charkhgard, Savelsbergh, and Talebian) in this study, to solve a relaxation of Problem (2.1). The relaxation can be obtained by dropping the integrality condition of binary and integer decision variables of Problem (2.1).

By definition, the optimal objective value of such a relaxation should provide a dual bound, i.e, an upper bound, for the optimal objective value of Problem (2.1). Therefore, by this observation, in this paper, we frequently use CST to compute dual bounds. However, whenever we use CST, we provide a lower bound vector, denoted by $\boldsymbol{l} \in \mathbb{R}^n$, and an upper bound vector, denoted by $\boldsymbol{u} \in \mathbb{R}^n$, for $\boldsymbol{x}$. As an aside, we note that $\boldsymbol{l}$ and $\boldsymbol{u}$ will be provided/updated automatically during the course of the branch-and-bound algorithm that we will introduce in the next section. So, we introduce the operation CST$(\boldsymbol{l}, \boldsymbol{u})$ which is equivalent to solving the following problem:

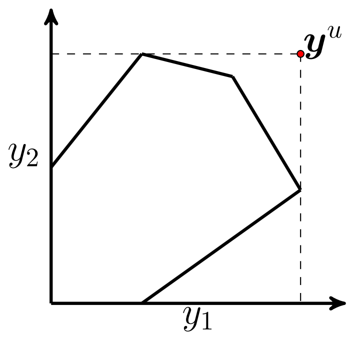
$$\begin{aligned} \max\ & \prod_{i=1}^{2} y_i \\ \text{s.t. } & \boldsymbol{y} = D\boldsymbol{x} + \boldsymbol{d} \\ & A\boldsymbol{x} \leqslant \boldsymbol{b} \\ & \boldsymbol{l} \leqslant \boldsymbol{x} \leqslant \boldsymbol{u} \\ & \boldsymbol{x}, \boldsymbol{y} \geqslant \boldsymbol{0}, \quad \boldsymbol{x} \in \mathbb{R}^n, \quad \boldsymbol{y} \in \mathbb{R}^2, \end{aligned} \tag{2.2}$$

by calling CST algorithm. This operation simply returns an optimal solution and an optimal point of Problem (2.2) denoted by $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$. Note that if $\tilde{\boldsymbol{x}}$ =null (or equivalently $\tilde{\boldsymbol{y}}$=null) then Problem (2.2) is infeasible. Next, we provide a high-level description of the CST algorithm. Interested readers can refer to [3] for further details.

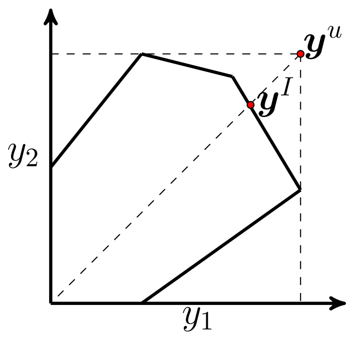
Suppose that CST$(\boldsymbol{l}, \boldsymbol{u})$ is called. An illustration of the feasible set of Problem (2.2) in the criterion space can be found in Figure 2.1a. In each iteration, CST computes a global upper bound point, denoted by $\boldsymbol{y}^u$, and a global lower bound point, denoted by $\boldsymbol{y}^l$, in the criterion space. This implies that $\boldsymbol{y}_1^u \boldsymbol{y}_2^u$ and $\boldsymbol{y}_1^l \boldsymbol{y}_2^l$ provide a global upper bound and a global lower bound for the optimal objective value of Problem (2.2), respectively. The algorithm terminates whenever the (relative and/or absolute) optimality gap falls below a given threshold. In the first iteration, the algorithm computes an upper bound point by solving two single-objective linear programs. The first maximizes y_1 and the second maximizes y_2 over the feasible set in the criterion space. An illustration of the upper bound point in the criterion space can be found in Figure 2.1b. Next, the algorithm searches over the imaginary line that passes through the upper bound point and the origin to compute the so-called intersection point which is denoted by $\boldsymbol{y}^I$. The intersection point is the closest feasible point in the criterion space to the upper bound point and can be computed by solving a single-objective linear program. Therefore, $y_1^I y_2^I$ provides a global lower bound for the optimal objective value of Problem (2.2). An illustration of the intersection point found



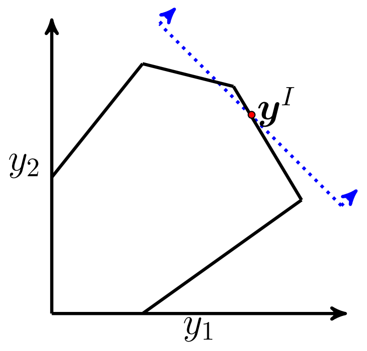
(a) The feasible set in the criterion space



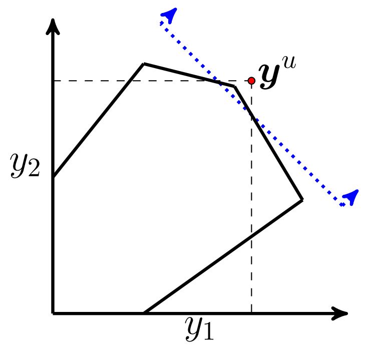
(b) The first upper bound point



(c) The first intersection point



(d) The first cut



(e) The second upper bound point

Figure 2.1: An illustration of the workings of CST($\boldsymbol{l}, \boldsymbol{u}$)

in the first iteration can be found in Figure 2.1c. Note that the algorithm keeps the best global lower bound found during the course of the algorithm in $\boldsymbol{y}^l$. In other words, after computing an intersection point, $\boldsymbol{y}^l$ should be updated. Next, the algorithm adds a cut to the criterion space based on the position of the intersection point to remove the parts of the criterion space that cannot contain a better lower bound point. An illustration of the cut added in the first iteration can be found in Figure 2.1d. After adding the cut, the next iteration starts and the same process repeats but this time over the reduced feasible set in the criterion space. For example, an illustration of the upper bound point produced in the second iteration can be found in Figure 2.1e.

Remark 2.1. For instances with $p > 2$, [3] have shown that CST can provide good feasible solutions with provable quality guarantee but the algorithm cannot necessarily converge to an optimal solution. So, further research is required about customizing CST for solving instances with $p > 2$.

2.2 A Branch-and-Bound Algorithm

Our proposed algorithm is similar to the traditional branch-and-bound algorithm for single-objective mixed integer linear programs. The main difference is that instead of using a linear programming solver to compute dual bounds, we employ the CST algorithm. Next, we provide a detailed explanation of our proposed branch-and-bound algorithm.

The algorithm maintains a queue of nodes, denoted by TREE. The algorithm also maintains a global upper bound, denoted by $\mathrm{G_{UB}}$, and global lower bound, denoted by $\mathrm{G_{LB}}$. At the beginning, the algorithm sets $\mathrm{G_{UB}} = +\infty$ and $\mathrm{G_{LB}} = -\infty$. Moreover, the algorithm initializes $(\boldsymbol{l}, \boldsymbol{u})$ with $(\boldsymbol{0}, +\infty)$. To initialize the queue, the algorithm first calls $\mathrm{CST}(\boldsymbol{l}, \boldsymbol{u})$ to compute $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$. If the integrality conditions hold, i.e., $\tilde{\boldsymbol{x}} \in \mathbb{R}^{n_c} \times \mathbb{B}^{n_b} \times \mathbb{Z}^{n_i}$, then an optimal solution is found. So, in that case, the algorithm terminates after setting $(\boldsymbol{x}^*, \boldsymbol{y}^*) = (\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$. Otherwise, $\mathrm{CST}(\boldsymbol{l}, \boldsymbol{u})$ has computed a dual bound, and so the algorithm sets $\mathrm{G_{UB}} = \tilde{y}_1 \tilde{y}_2$ and initializes the queue by $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}, \boldsymbol{l}, \boldsymbol{u})$. The algorithm then explores the queue as long as it is nonempty and $\mathrm{G_{UB}} - \mathrm{G_{LB}} \geqslant \varepsilon_1$ and $\frac{\mathrm{G_{UB}}-\mathrm{G_{LB}}}{\mathrm{G_{UB}}} \geqslant \varepsilon_2$, where $\varepsilon_1 > 0$ and $\varepsilon_2 \in (0, 1)$ are the user-defined absolute and relative optimality gap tolerances, respectively. Next, we explain how each element of the queue is explored.

In each iteration, the algorithm pops out an element of the queue and denotes it by $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{u})$. Note that when an element is popped out from the queue then that element does not exist in the queue anymore. We will explain in detail in which order the elements of the queue should be popped out in Section 2.4. The algorithm next selects an index of a decision variable that was supposed to take an integer (or binary) value but it currently has fractional value in solution $\boldsymbol{x}$. This operation is denoted by Branching-Index($\boldsymbol{x}$) and its output is denoted by j. In Section 2.3, we will introduce several rules for this operation.

Next, the algorithm generates two new lower bound and upper bound vectors, denoted by $(\boldsymbol{l}^1, \boldsymbol{u}^1)$ and $(\boldsymbol{l}^2, \boldsymbol{u}^2)$. The algorithm sets $l_i^1 = l_i$ for all $i \in \{1, \ldots, n\} \backslash \{j\}$, $l_j^1 = \lceil x_j \rceil$, and $\boldsymbol{u}^1 = \boldsymbol{u}$. Similarly, the algorithm sets $u_i^2 = u_i$ for all $i \in \{1, \ldots, n\} \backslash \{j\}$, $u_j^2 = \lfloor x_j \rfloor$, and $\boldsymbol{l}^2 = \boldsymbol{l}$. The algorithm first explores $(\boldsymbol{l}^1, \boldsymbol{u}^1)$. This implies that the algorithm calls $\mathrm{CST}(\boldsymbol{l}^1, \boldsymbol{u}^1)$ to compute $(\boldsymbol{x}^1, \boldsymbol{y}^1)$. Again we note that if CST fails to find a feasible solution then we have that $\boldsymbol{x}^1$ =null and $\boldsymbol{y}^1$ =null. Therefore, in that case, the algorithm will skip the following steps and proceed to explore $(\boldsymbol{l}^2, \boldsymbol{u}^2)$. Otherwise, the algorithm first checks whether the integrality conditions hold, i.e., $\boldsymbol{x}^1 \in \mathbb{R}^{n_c} \times \mathbb{B}^{n_b} \times \mathbb{Z}^{n_i}$, and $\mathrm{G_{LB}} < y_1^1 y_2^1$. If that is the case then a new and better global lower bound is found and so the algorithm will set $(\boldsymbol{x}^*, \boldsymbol{y}^*) = (\tilde{\boldsymbol{x}}^1, \tilde{\boldsymbol{y}}^1)$ and $\mathrm{G_{LB}} = y_1^1 y_2^1$. Otherwise, the algorithm checks whether $y_1^1 y_2^1 - \mathrm{G_{LB}} \geqslant \varepsilon_1$ and $\frac{y_1^1 y_2^1 - \mathrm{G_{LB}}}{y_1^1 y_2^1} \geqslant \varepsilon_2$. If that is the case then the algorithm will add $(\boldsymbol{x}^1, \boldsymbol{y}^1, \boldsymbol{l}^1, \boldsymbol{u}^1)$ to be explored further in the future because it is possible to find better feasible solutions by exploring that element.

After exploring $(\boldsymbol{l}^1, \boldsymbol{u}^1)$, $(\boldsymbol{l}^2, \boldsymbol{u}^2)$ should be explored similarly. Therefore, the algorithm will explore it and then before starting the next iteration it will update $\mathrm{G_{UB}}$. The maximum value of $y_1 y_2$ among all nodes in the queue defines the new global upper bound. A detailed description of the proposed branch-and-bound algorithm can be found in Algorithm 1.

```
Input: A feasible instance of Problem (2.1)
$Queue.create(\textsc{Tree})$
$(\boldsymbol{l}, \boldsymbol{u}) \leftarrow (\boldsymbol{0}, +\infty)$
$\mathrm{G_{LB}} \leftarrow -\infty$; $\mathrm{G_{UB}} \leftarrow +\infty$
$(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) \leftarrow \mathrm{CST}(\boldsymbol{l}, \boldsymbol{u})$
if $\tilde{\boldsymbol{x}} \in \mathbb{R}^{n_c} \times \mathbb{B}^{n_b} \times \mathbb{Z}^{n_i}$ then
    $(\boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow (\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$
    $\mathrm{G_{LB}} \leftarrow \tilde{y}_1\tilde{y}_2$
end
else
    $\textsc{Tree}.add\big((\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}, \boldsymbol{l}, \boldsymbol{u})\big)$
    $\mathrm{G_{UB}} \leftarrow \tilde{y}_1\tilde{y}_2$
end
while not $Queue.empty(\textsc{Tree})$ & $\mathrm{G_{UB}} - \mathrm{G_{LB}} \geqslant \varepsilon_1$ & $\frac{\mathrm{G_{UB}} - \mathrm{G_{LB}}}{\mathrm{G_{UB}}} \geqslant \varepsilon_2$ do
    $\textsc{Tree}.PopOut\big((\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{u})\big)$
    $j \leftarrow \textsc{Branching-Index}(\boldsymbol{x})$
    $(\boldsymbol{l}^1, \boldsymbol{u}^1) \leftarrow (\boldsymbol{l}, \boldsymbol{u})$; $l_j^1 \leftarrow \lceil x_j \rceil$
    $(\boldsymbol{l}^2, \boldsymbol{u}^2) \leftarrow (\boldsymbol{l}, \boldsymbol{u})$; $u_j^2 \leftarrow \lfloor x_j \rfloor$
    $(\boldsymbol{x}^1, \boldsymbol{y}^1) \leftarrow \mathrm{CST}(\boldsymbol{l}^1, \boldsymbol{u}^1)$
    if $(\boldsymbol{x}^1, \boldsymbol{y}^1) \neq$ (null,null) then
        if $\boldsymbol{x}^1 \in \mathbb{R}^{n_c} \times \mathbb{B}^{n_b} \times \mathbb{Z}^{n_i}$ & $y_1^1 y_2^1 > \mathrm{G_{LB}}$ then
            $(\boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow (\boldsymbol{x}^1, \boldsymbol{y}^1)$
            $\mathrm{G_{LB}} \leftarrow y_1^1 y_2^1$
        end
        else if $y_1^1 y_2^1 - \mathrm{G_{LB}} \geqslant \varepsilon_1$ & $\frac{y_1^1 y_2^1 - \mathrm{G_{LB}}}{y_1^1 y_2^1} \geqslant \varepsilon_2$ then
            $\textsc{Tree}.add\big((\boldsymbol{x}^1, \boldsymbol{y}^1, \boldsymbol{l}^1, \boldsymbol{u}^1)\big)$
        end
    end
    $(\boldsymbol{x}^2, \boldsymbol{y}^2) \leftarrow \mathrm{CST}(\boldsymbol{l}^2, \boldsymbol{u}^2)$
    if $(\boldsymbol{x}^2, \boldsymbol{y}^2) \neq$ (null,null) then
        if $\boldsymbol{x}^2 \in \mathbb{R}^{n_c} \times \mathbb{B}^{n_b} \times \mathbb{Z}^{n_i}$ & $y_1^2 y_2^2 > \mathrm{G_{LB}}$ then
            $(\boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow (\boldsymbol{x}^2, \boldsymbol{y}^2)$
            $\mathrm{G_{LB}} \leftarrow y_1^2 y_2^2$
        end
        else if $y_2^2 y_2^2 - \mathrm{G_{LB}} \geqslant \varepsilon_1$ & $\frac{y_1^2 y_2^2 - \mathrm{G_{LB}}}{y_1^2 y_2^2} \geqslant \varepsilon_2$ then
            $\textsc{Tree}.add\big((\boldsymbol{x}^2, \boldsymbol{y}^2, \boldsymbol{l}^2, \boldsymbol{u}^2)\big)$
        end
    end
    Update $\mathrm{G_{UB}}$
end
return $\boldsymbol{x}^*, \boldsymbol{y}^*$
```

Algorithm 1: A Branch-and-Bound Algorithm

2.3 Branching Strategies

Selecting a branching variable is a crucial task in any branch-and-bound algorithm since it can impact the solution time of the algorithm significantly. Ideally, we would like to select a variable for branching that helps us to explore the minimum number of nodes in total. In light of this observation, in this study, we explore some variants of several well-known branching strategies that will define the operation BRANCHING-INDEX($\boldsymbol{x}$) in Algorithm 1. Let I be the index sets of all binary and integer decision variables in Problem (2.1). For a given node of the queue $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{u})$, let $V(\boldsymbol{x}) \subseteq I$ be the index set of all variables in solution $\boldsymbol{x}$ that were supposed to take integer (or binary) values but they have taken fractional values. We have explored the following five branching strategies in this study (interested readers can refer to 40 for further details):

- *Random branching*: This is the easiest and the least-memory consuming branching rule. In this rule, for a given node of the queue $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{u})$, all the elements of $V(\boldsymbol{x})$ have an equal chance of being selected, and so the algorithm selects one of them randomly.

- *Most infeasible branching*: For a given node of the queue $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{u})$, another simple branching strategy is to randomly select an element of $V(\boldsymbol{x})$ with the largest integer violation for branching. For each $i \in V(\boldsymbol{x})$, the integer violation is defined as $\min(x_i - \lfloor x_i \rfloor, \lceil x_i \rceil - x_i)$.

  In the traditional branch-and-bound algorithm for single-objective mixed integer linear programming, it is known that this branching rule does not outperform random branching [41]. In fact, this rule normally has a poor performance. However, surprisingly, we will computationally show (in Section 2.6) that this rule performs the best for our proposed branch-and-bound algorithm for solving Problem (2.1).

- *Pseudo-costs branching*: This strategy was initially introduced by [42] and then explored further by [43]. This strategy maintains a history of the results of past branchings for each $i \in I$. For a given node of the queue $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{u})$, the algorithm selects an index $i \in V(\boldsymbol{x})$ with the largest expected change in the dual bound, i.e., upper bound. Next, we explain how pseudo-costs can be computed for each iteration.

Let ρ_i^1 and ρ_i^2 be the pseudo-costs for $i \in I$ at any time during the course of the algorithm. For each $i \in I$, at the beginning, the algorithm sets $\rho_i^1 = 0$ and $\rho_i^2 = 0$. In any iteration of the algorithm, if the algorithm branches on $i \in I$ and calls $\mathrm{CST}(\boldsymbol{l}^1, \boldsymbol{u}^1)$, the algorithm updates ρ_i^1 as follows:

$$\rho_i^1 \leftarrow \rho_i^1 + \frac{y_1y_2 - y_1^1y_2^1}{\lceil x_i \rceil - x_i},$$

where y_1y_2 is the optimal objective value associated with the node that the algorithm is exploring at that iteration, i.e., $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{u})$. We denote the number of times that ρ_i^1 has been updated by n_i^1 at any time during the course of the algorithm. Note that if after calling $\mathrm{CST}(\boldsymbol{l}^1, \boldsymbol{u}^1)$ it turns out that $(\boldsymbol{x}^1, \boldsymbol{y}^1)$ =(null,null), i.e., meaning the problem is infeasible, then ρ_i^1 and n_i^1 should not be updated. Similarly, whenever the algorithm branches on $i \in I$ and calls $\mathrm{CST}(\boldsymbol{l}^2, \boldsymbol{u}^2)$, the algorithm updates ρ_i^2 as follows:

$$\rho_i^2 \leftarrow \rho_i^2 + \frac{y_1y_2 - y_1^2y_2^2}{x_i - \lfloor x_i \rfloor}.$$

We denote the number of times that ρ_i^2 has been updated by n_i^2 at any time during the course of the algorithm. Note that if after calling $\mathrm{CST}(\boldsymbol{l}^2, \boldsymbol{u}^2)$ it turns out that $(\boldsymbol{x}^2, \boldsymbol{y}^2)$ =(null,null), i.e., meaning the problem is infeasible, then ρ_i^2 and n_i^2 should not be updated.

In light of the above, suppose that at a particular iteration of the algorithm, we are exploring the node $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{u})$ and we want to decide on which variable we should branch. Based on the pseudo-costs, one can estimate the change that can occur in the dual bound by calling $\text{CST}(\boldsymbol{l}^1, \boldsymbol{u}^1)$ after branching on $i \in V(\boldsymbol{x})$ as follows:

$$\delta_i^1 := (\lceil x_i \rceil - x_i)\frac{\rho_i^1}{n_i^1}.$$

In other words, we estimate that $y_1 y_2 - y_1^1 y_2^1 \approx \delta_i^1$. Similarly, one can estimate the change that can occur in the dual bound by calling $\text{CST}(\boldsymbol{l}^2, \boldsymbol{u}^2)$ after branching on $i \in V(\boldsymbol{x})$ as follows:

$$\delta_i^2 := (x_i - \lfloor x_i \rfloor)\frac{\rho_i^2}{n_i^2}$$

In other words, we estimate that $y_1 y_2 - y_1^2 y_2^2 \approx \delta_i^2$. Therefore, the algorithm can compute a score for each $i \in V(\boldsymbol{x})$ as follows:

$$s_i := \mu \min(\delta_i^1, \delta_i^2) + (1 - \mu) \max(\delta_i^1, \delta_i^2)$$

where $\mu \in [0, 1]$ is a user-defined parameter which is typically close to 1. During the course of this study, we examined different values for μ, i.e., $\{0, 0.1, 0.2, \ldots, 1\}$. However, we set $\mu = 0.9$ in this paper because this value performs the best. The index with the highest score is the one that the algorithm selects for branching. In order to employ this sophisticated rule, an initial estimation is required. In this paper, we use the following strategies to obtain an initial estimation.

1. Initialization using the random branching: In this strategy, we employ random branching for a number of iterations (which is 10 in this paper) and then the algorithm switches to pseudo-costs branching. It is worth mentioning that if at a particular iteration the pseudo-costs branching fails to select an index, i.e., $s_i = 0$ for all $i \in V(\boldsymbol{x})$, then we employ the random branching for that particular iteration.

2. Initialization using the most infeasible branching: This is similar to the previous case, we only employ the most infeasible branching instead of the random branching.

- *Reliability branching:* This branching is mostly based on the *strong* branching [44]. Suppose that at a particular iteration of the algorithm, we are exploring the node $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{u})$ and we want to decide on which variable we should branch using the strong branching. In strong branching, the algorithm will actually call both $\mathrm{CST}(\boldsymbol{l}^1, \boldsymbol{u}^1)$ and $\mathrm{CST}(\boldsymbol{l}^2, \boldsymbol{u}^2)$ to compute the exact change in the dual bound, i.e., $\Delta_i^1 := y_1 y_2 - y_1^1 y_2^1$ and $\Delta_i^2 := y_1 y_2 - y_1^2 y_2^2$. So, for each $i \in V(\boldsymbol{x})$, the algorithm computes s_i but using Δ_i^1 and Δ_i^2 instead of δ_i^1 and δ_i^2. Similar to the pseudo-costs branching, the index with the highest score is the one that the algorithm selects for branching. Since strong branching imposes a high computational burden, [41] suggest that we should combine the strong branching with the pseudocosts branching. This is known as the *reliability branching*. If $n_i^1 \leqslant \tau$ then we use Δ_i^1 in computing s_i, and otherwise we use δ_i^1. Also, if $n_i^2 \leqslant \tau$ then we use Δ_i^2 in computing s_i, and otherwise we use δ_i^2. During the course of this study, we examined different values for τ, i.e., $\{1, 2, 4, 8, 16, 32\}$. However, we set $\tau = 2$ in this paper because this value performs the best.

2.4 Node Selecting Strategies

Selecting which node should be popped out in each iteration is another crucial task in any branch-and-bound algorithm since it can impact the solution time of the algorithm significantly. The aim of node selecting strategies is pruning open nodes and ending the queue as quickly as possible. This can be done by either finding a good feasible solution, in order to increase the global lower bound (or primal bound), or decreasing the global upper bound (or dual bound). We implemented five different node selecting strategies, two of which are solely based on pseudocosts branching [45, 46]:

- *Depth-first search*: In this strategy, the node most recently added to the tree is chosen for branching [47]. The two advantages of this strategy are that, if there is no primary feasible solution on hand, this strategy finds one quickly, and by doing so, it increases G_{LB} quickly. The other advantage is that it keeps the list of open nodes minimal, and therefore, the memory-usage is small. However, on the negative side, this strategy is slow in improving the global dual bound and so it takes a lot of time to prove the optimality of a solution.

- *Best-bound search*: This strategy selects the node with the best upper/dual bound [48]. Although this strategy needs more memory in comparison to other strategies, if an optimal solution is available, this strategy is the fastest to prove its optimality. Another important characteristic of this strategy is that, in practice, the first feasible solution that the algorithm finds under this strategy is usually an optimal solution of the problem.

- *Two-phase method*: The idea of this strategy is to combine depth-first search and best-bound search strategies to use the benefits of both. In this paper, our proposed algorithm employs the depth-first search strategy for at most $\min(1000, 3n)$ number of iterations. As soon as a feasible solution is found which is better than the current G_{LB} or the algorithm reaches to the maximum number of iterations then it switches to the best-bound search. The algorithm operates on the best-bound search for $\min(1000, 3n)$ number of iterations. As soon as the algorithm reaches to its maximum number of iterations then the algorithm checks how much the global upper bound has improved during operating on the best-bound search strategy. If the improvement is greater than or equal to 5% then the algorithm starts to operate on the best-bound search for $\min(1000, 3n)$ number of iterations again. Otherwise, it will return to the depth-first search strategy and similar procedures discussed above will be repeated.

- *Best expected bound*: This strategy is based on the pseudo-costs and selects the node with the best expected dual bound after branching. For any given node $N := (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{u})$ of the queue, let $\delta^1(N)$ and $\delta^2(N)$ be the change estimate in the dual bound computed by the pseudo-costs branching if we branch on the variable that pseudo-costs branching suggests for that particular node. So, in that case, the estimated dual bounds will be for node $N \in \textsc{Tree}$:

  $$\bar{e}_N^1 := y_1 y_2 - \delta^1(N),$$

  and

  $$\bar{e}_N^2 := y_1 y_2 - \delta^2(N).$$

  In light of the above, in this strategy, the algorithm selects a node $N \in \textsc{Tree}$ that has the largest value of $\max\{\bar{e}_N^1, \bar{e}_N^2\}$.

- *Best estimate*: This strategy is also based on the pseudo-costs. This strategy selects a node that is expected to result in the best integer solution (or best primal bound). The best expected primal bound for a given node $N := (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{u})$ of the queue can be computed as follows:

  $$\underline{e}_N := y_1 y_2 - \sum_{i \in V(\boldsymbol{x})} \min\{\delta_i^1, \delta_i^2\}$$

  In light of the above, in this strategy, the algorithm selects a node $N \in \textsc{Tree}$ that has the largest value of $\underline{e}_N$.

2.5 Enhancements

In this section, we explain a preprocessing technique which is developed for the aim of producing good global lower bounds and cuts for Problem (2.1). The preprocessing technique can be called before running Algorithm 1 to possibly improve the performance of this algorithm. Obviously, generating a good global lower/primal bound can be helpful since the nodes of the branch-and-bound tree can be pruned faster. Also, by generating cuts and adding them to the formulation, better global upper/dual bounds can be computed during the course of Algorithm 1.

The proposed preprocessing technique is developed based on Proposition 2.1. By this proposition, any efficient solution is expected to be a high-quality feasible solution and hence it can be considered as a (good) global lower bound for the problem. Therefore, in the proposed preprocessing technique, we attempt to enumerate some of the efficient solutions of the problem. In order to do so, we use the weighted sum operation, denoted by WSO(λ_1, λ_2):

$$\begin{aligned}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \in \arg\max \{&\lambda_1 y_1 + \lambda_2 y_2 : \\ &\boldsymbol{y} = D\boldsymbol{x} + \boldsymbol{d},\ A\boldsymbol{x} \leqslant \boldsymbol{b},\ \boldsymbol{x} \geqslant \boldsymbol{0},\ \boldsymbol{y} \geqslant \boldsymbol{0},\ \boldsymbol{x} \in \mathbb{R}^{n_c} \times \mathbb{B}^{n_b} \times \mathbb{Z}^{n_i},\ \boldsymbol{y} \in \mathbb{R}^2\},\end{aligned}$$

where $\lambda_1, \lambda_2 > 0$ are user-defined weights. In multi-objective optimization, it is known that this operation always returns an efficient solution (if there exists any) for a multi-objective optimization problem. However, for non-convex optimization problems such as Problem (2.1), not all efficient solutions can be computed using this operation by considering even all possible weights [49]. In addition to the fact that $\bar{y}_1\bar{y}_2$ provides a global lower bound, the weighted sum operation can naturally produce the following cut for Problem (2.1), due to optimality:

$$\lambda_1 y_1 + \lambda_2 y_2 \leqslant \lambda_1 \bar{y}_1 + \lambda_2 \bar{y}_2.$$

In light of the above, in the proposed preprocessing technique, we employ the weighted sum operation to generate a feasible solution (i.e., global primal bound) and a cut for the problem. In order to use the weighted sum operation effectively, we employ the *Weighted Sum Method (WSM)* [50]. This is because the WSM can compute all nondominated points that are extreme points of the convex hull of $\mathcal{Y}$. Next, we explain a high-level description of the WSM.

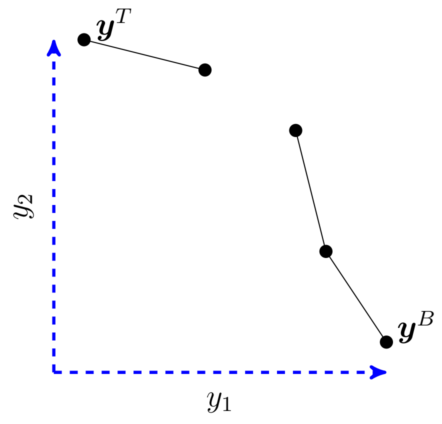
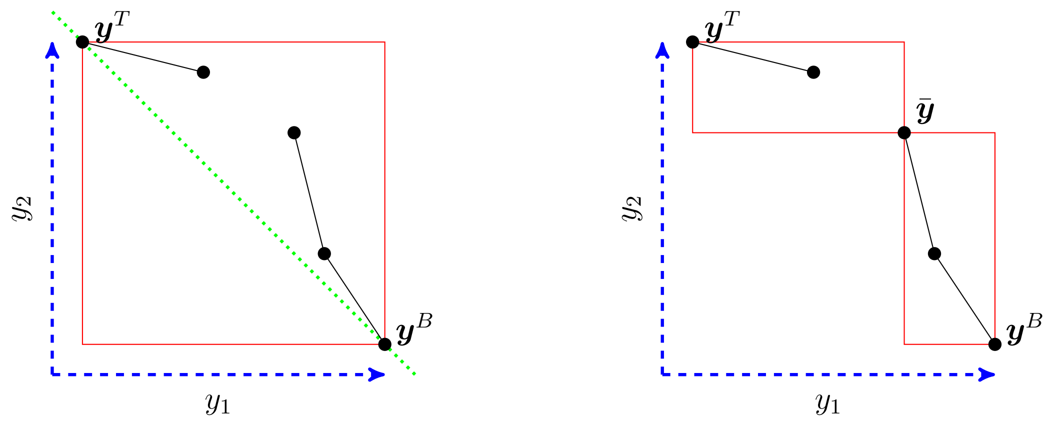


Figure 2.2: An illustration of the nondominated frontier corresponding to Problem (2.1)



(a) $R(\boldsymbol{y}^T, \boldsymbol{y}^B)$ and the new function

(b) The new imaginary rectangles

Figure 2.3: An illustration of the workings of the weighted sum method

In the WSM, we first compute the endpoints of the nondominated frontier. The top endpoint, denoted by $\boldsymbol{y}^T$, can be computed by first solving,

$$(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) \in \arg\max \{y_2 : \\ \boldsymbol{y} = D\boldsymbol{x} + \boldsymbol{d},\ A\boldsymbol{x} \leqslant \boldsymbol{b},\ \boldsymbol{x} \geqslant \boldsymbol{0},\ \boldsymbol{y} \geqslant \boldsymbol{0},\ \boldsymbol{x} \in \mathbb{R}^{n_c} \times \mathbb{B}^{n_b} \times \mathbb{Z}^{n_i},\ \boldsymbol{y} \in \mathbb{R}^2\},$$

and if it is feasible, it needs to be followed by solving

$$(\boldsymbol{x}^T, \boldsymbol{y}^T) \in \arg\max \{y_1 :\ y_2 \geqslant \tilde{y}_2 \\ \boldsymbol{y} = D\boldsymbol{x} + \boldsymbol{d},\ A\boldsymbol{x} \leqslant \boldsymbol{b},\ \boldsymbol{x} \geqslant \boldsymbol{0},\ \boldsymbol{y} \geqslant \boldsymbol{0},\ \boldsymbol{x} \in \mathbb{R}^{n_c} \times \mathbb{B}^{n_b} \times \mathbb{Z}^{n_i},\ \boldsymbol{y} \in \mathbb{R}^2\},$$

We denote the operation of computing $(\boldsymbol{x}^T, \boldsymbol{y}^T)$ by $\arg \operatorname{lex} \max(y_2, y_1)$. Similarly, the bottom endpoint can be computed by using $\arg \operatorname{lex} \max(y_1, y_2)$ and its outcome is denoted by $(\boldsymbol{x}^B, \boldsymbol{y}^B)$. An illustration of the nondominated frontier corresponding to Problem (2.1) can be found in Figure 2.2. It is evident that if $(\boldsymbol{x}^T, \boldsymbol{y}^T) =$(null,null), i.e., Problem (2.1) is infeasible, then we must have that $(\boldsymbol{x}^B, \boldsymbol{y}^B) =$(null,null). Therefore, there is no need to compute $(\boldsymbol{x}^B, \boldsymbol{y}^B)$ in that case. Also, if $\boldsymbol{y}^T = \boldsymbol{y}^B$ then the Problem (2.1) has an *ideal* point, i.e., $|\mathcal{Y}_N| = 1$. In the remaining, we assume that $\boldsymbol{y}^T \neq \boldsymbol{y}^B$.

At the beginning of the WSM, we set $\mathrm{G_{LB}} = -\infty$. After computing the endpoints of the nondominated frontier, both $y_1^T y_2^T$ and $y_1^B y_2^B$ are global lower bounds for Problem (2.1). So, we set $\mathrm{G_{LB}}$ to the best/highest lower bound found and update $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ accordingly. Moreover, we know that $y_1 \leqslant y_1^B$ and $y_2 \leqslant y_2^T$ are two valid cuts that can be added to the formulation and so we add them to the list of cuts which is denoted by CUTS. Note that after computing the endpoints, all other nondominated points must be in the imaginary rectangle defined by $\boldsymbol{y}^T$ and $\boldsymbol{y}^B$, denoted by $R(\boldsymbol{y}^T, \boldsymbol{y}^B)$ (See Figure 2.3a). Hence, the WSM explores this rectangle and may change it into smaller rectangles, and repeat this process in each one until it finds all extreme nondominated points.

```
Input: An instance of Problem (2.1)
$List.create(\textsc{Cuts})$
$Queue.create(Q)$
$\mathrm{G_{LB}} = -\infty$
$(\boldsymbol{x}^T, \boldsymbol{y}^T) \leftarrow \arg\operatorname{lex}\max(y_2, y_1)$
if $(\boldsymbol{x}^T, \boldsymbol{y}^T) \neq (null, null)$ then
    $\boldsymbol{y}^B \leftarrow \arg\operatorname{lex}\max(y_1, y_2)$
    $\mathrm{G_{LB}} \leftarrow y_1^T y_2^T$, $(\boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow (\boldsymbol{x}^T, \boldsymbol{y}^T)$
    $\textsc{Cuts}.add(y_2 \leqslant y_2^T)$
    if $\boldsymbol{y}^T \neq \boldsymbol{y}^B$ then
        $Q.add(\boldsymbol{R}(\boldsymbol{y}^T, \boldsymbol{y}^B))$
        if $\mathrm{G_{LB}} < y_1^B y_2^B$ then
            $\mathrm{G_{LB}} \leftarrow y_1^B y_2^B$, $(\boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow (\boldsymbol{x}^B, \boldsymbol{y}^B)$
        end
        $\textsc{Cuts}.add(y_1 \leqslant y_1^B)$
    end
end
while $not\ Queue.empty(\textsc{Tree})$ do
    $Q.PopOut(\boldsymbol{R}(\boldsymbol{y}^1, \boldsymbol{y}^2))$
    $\lambda_1 \leftarrow y_2^1 - y_2^2$
    $\lambda_2 \leftarrow y_1^2 - y_1^1$
    $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \leftarrow \mathrm{WSO}(\lambda_1, \lambda_2)$
    if $\mathrm{G_{LB}} < \bar{y}_1 \bar{y}_2$ then
        $\mathrm{G_{LB}} \leftarrow \bar{y}_1 \bar{y}_2$, $(\boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow (\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$
    end
    $\textsc{Cuts}.add(\lambda_1 y_1 + \lambda_2 y_2 \leqslant \lambda_1 \bar{y}_1 + \lambda_2 \bar{y}_2)$
    if $\lambda_1 \bar{y}_1 + \lambda_2 \bar{y}_2 > \lambda_1 y_1^1 + \lambda_2 y_2^1$ then
        $Q.add(\boldsymbol{R}(\boldsymbol{y}^1, \bar{\boldsymbol{y}}))$
        $Q.add(\boldsymbol{R}(\bar{\boldsymbol{y}}, \boldsymbol{y}^2))$
    end
end
return $\boldsymbol{x}^*, \boldsymbol{y}^*, \mathrm{G_{LB}}, \textsc{Cuts}$
```

Algorithm 2: Preprocessing

More precisely, to explore a given rectangle $R(\boldsymbol{y}^1, \boldsymbol{y}^2)$ with $y_1^1 < y_1^2$ and $y_2^1 > y_2^2$, the WSM calls $\text{WSO}(\lambda_1, \lambda_2)$ after setting $\lambda_1 = y_2^1 - y_2^2$ and $\lambda_2 = y_1^2 - y_1^1$ to compute $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$. Note that in this way, $\lambda_1 y_1 + \lambda_2 y_2$ is a function which is parallel to the line that connects y^1 and y^2 in the criterion space. Figure 2.3a shows an example with $y^1 = y^T$ and $y^2 = y^B$. As discussed before since $\bar{\boldsymbol{y}}$ is a nondominated point, if $\text{G}_{\text{LB}} < \bar{y}_1\bar{y}_2$ then we set $\text{G}_{\text{LB}} = \bar{y}_1\bar{y}_2$ and $(\boldsymbol{x}^*, \boldsymbol{y}^*) = (\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$. Also, the WSM adds $\lambda_1 y_1 + \lambda_2 y_2 \leqslant \lambda_1 \bar{y}_1 + \lambda_2 \bar{y}_2$ to the list of cuts. Finally, it is evident that if $\lambda_1 \bar{y}_1 + \lambda_2 \bar{y}_2 > \lambda_1 y_1^1 + \lambda_2 y_2^1$ then $\bar{\boldsymbol{y}}$ is not a convex combination of $\boldsymbol{y}^1$ and $\boldsymbol{y}^2$. So, in that case, a similar procedure is applied recursively to search $R(\boldsymbol{y}^1, \bar{\boldsymbol{y}})$ and $R(\bar{\boldsymbol{y}}, \boldsymbol{y}^2)$ for additional nondominated points (see Figure 2.3b). Algorithm 2 shows a precise description of the proposed preprocessing technique.

2.6 Computational Study

As we will explain later in this section, based on the study of [51], Problem (2.1) can be reformulated as a mixed integer SOCP. So, in this section, we compare the performance of our proposed branch-and-bound algorithm with the performance of the mixed integer SOCP solver of CPLEX 12.7 (CPLEX-MI-SOCP). We implement our algorithm in C++ and use CPLEX 12.7 to solve linear programs and mixed integer programs arising during the course of Algorithm 1 and Algorithm 2. 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, ε_1, and the relative optimality gap, ε_2, are set to 10^{-6} in both solution methods, i.e., our branch-and-bound algorithm and CPLEX-MI-SOCP. Also, a time limit of 7,200 seconds is imposed for each instance in both solution methods. Furthermore, if the preprocessing is active then a time limit of $0.1n$ seconds is imposed for it. Note that the preprocessing can be used both in our proposed branch-and-bound algorithm and CPLEX-MI-SOCP. Consequently, if the preprocessing is active then a time limit of $7200 - t$ seconds is imposed for a solution method where t is the time (in seconds) used by the preprocessing (note that $t \leqslant 0.1n$). The code and instances used in this study can all be found at https://github.com/paymanghasemi, respectively. Next, we explain how the instances are generated.

Since pure integer instances can be linearized easily by introducing additional sets of variables and constraints, they can be solved as pure integer linear programs by commercial solvers. Consequently, in this computational study, all instances involve continuous decision variables, i.e., $n_c > 0$ for all instances. In particular, this computational study is conducted over two sets of instances each with 80 randomly generated instances. In both sets, $n_c = 0.5n$ for all instances. However, in the first set, there are no general integer variables, i.e, $n_i = 0$ and $n_b = 0.5n$, and in the second set, there is no binary decision variable, i.e., i.e, $n_b = 0$ and $n_i = 0.5n$. Each set contains 16 subclasses of instances based on the dimensions of the matrix $A_{m \times n}$, and each subclass contains 5 instances. Specifically, we assume that $m \in \{200, 400, 800, 1600\}$ and $n = \alpha m$ where $\alpha \in \{0.5, 1, 1.5, 2\}$. For example, the subclass 200×100 implies that $m = 200$ and $n = 100$, i.e., $\alpha = 0.5$. The sparsity of matrix A is set to 50%. The components of vector $\boldsymbol{b}$ and the entries of matrix A are randomly drawn from discrete uniform distributions $[50, 200]$ and $[10, 30]$, respectively. We set the components of vector $\boldsymbol{d}$ to zero. The sparsity of each row of the matrix D was also set to 50% and its components were drawn randomly from a discrete uniform distribution $[1, 10]$. Note that, since all constraints of the set $\mathcal{X}$ are inequality constraints and all coefficients of matrix A are nonnegative, the set $\mathcal{X}$ is bounded. Next, we explain how Problem (2.1) can be reformulated as mixed integer SOCP.

By introducing a new non-negative variable γ and introducing a *geometric mean* constraint, Problem (2.1) can be reformulated as follows:

$$\begin{aligned} &\max \gamma \\ &\text{s.t. } 0 \leqslant \gamma \leqslant \Big(\prod_{i=1}^{p} y_i\Big)^{\frac{1}{p}} \\ &\quad \boldsymbol{y} = D\boldsymbol{x} + \boldsymbol{d} \end{aligned}$$

$$A\boldsymbol{x} \leqslant \boldsymbol{b}$$

$$\boldsymbol{x}, \boldsymbol{y} \geqslant \boldsymbol{0}, \quad \boldsymbol{x} \in \mathbb{R}^{n_c} \times \mathbb{B}^{n_b} \times \mathbb{Z}^{n_i}, \quad \boldsymbol{y} \in \mathbb{R}^p, .$$

If $\bar{\gamma}$ is the optimal objective value of the reformulated problem, then $\bar{\gamma}^p$ is the optimal objective value of the original formulation. [51] show how an optimization problem of this form can be written as a mixed integer SOCP. Let k be the smallest integer value such that $2^k \geqslant p$. By introducing a set of non-negative variables and constraints, the geometric mean constraint can be replaced as follows:

$$\begin{aligned}
&\max \gamma \\
&\quad \text{s.t. } 0 \leqslant \gamma \leqslant \Gamma \\
&\qquad 0 \leqslant \Gamma \leqslant \sqrt{\tau_1^{k-1}\tau_2^{k-1}} \\
&\qquad 0 \leqslant \tau_j^l \leqslant \sqrt{\tau_{2j-1}^{l-1}\tau_{2j}^{l-1}} && \text{for } j = 1, \ldots, 2^{k-l} \text{ and } l = 1, \ldots, k-1 \\
&\qquad 0 \leqslant \tau_j^0 = y_j && \text{for } j = 1, \ldots, p \\
&\qquad 0 \leqslant \tau_j^0 = \Gamma && \text{for } j = p+1, \ldots, 2^k \\
&\qquad \boldsymbol{y} = D\boldsymbol{x} + \boldsymbol{d} \\
&\qquad A\boldsymbol{x} \leqslant \boldsymbol{b} \\
&\qquad \boldsymbol{x}, \boldsymbol{y} \geqslant \boldsymbol{0}, \quad \boldsymbol{x} \in \mathbb{R}^{n_c} \times \mathbb{B}^{n_b} \times \mathbb{Z}^{n_i}, \quad \boldsymbol{y} \in \mathbb{R}^p.
\end{aligned}$$

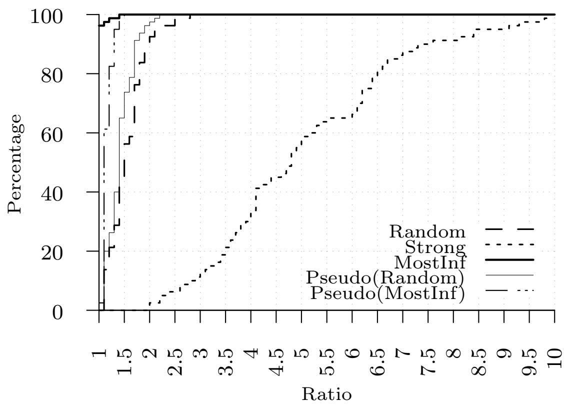
The above formulation is a mixed integer SOCP since any constraint of the form $\{u, v, w \geqslant 0 : u \leqslant \sqrt{vw}\}$ is equivalent to $\{u, v, w \geqslant 0 : \sqrt{u^2 + (\frac{v-w}{2})^2} \leqslant \frac{v+w}{2}\}$.

2.6.1 A Performance Comparison on Instance of Set I: Mixed Binary Instances

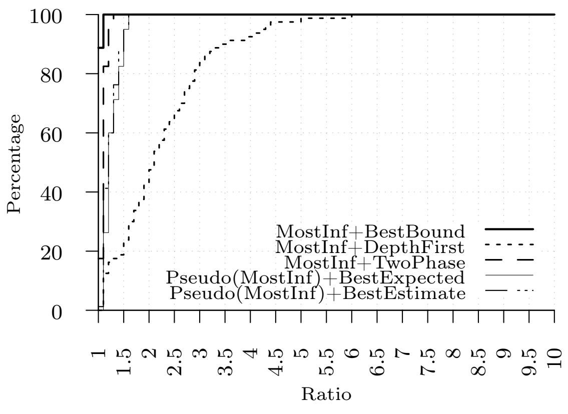
In this section, we first compare the performance of the proposed branch-and-bound algorithm under different settings obtained by employing the proposed branching strategies, node selecting strategies, and the preprocessing technique on the instances of set I, i.e., those with no general integer variables. After computing a good setting for the proposed algorithm, we then try to find a good setting for the CPLEX-MI-SOCP by exploring whether the proposed preprocessing technique is useful or not. Finally, we compare the performance of both solution methods under their best obtained settings.

It is worth mentioning that in this computational study, we frequently use *performance profiles* [52] to compare different algorithms in term of their solution times. A performance profile presents cumulative distribution functions for a set of algorithms being compared with respect to a specific performance metric. The run time performance profile for a set of algorithms is constructed by computing for each algorithm and for each instance the ratio of the run time of the algorithm on the instance and the minimum of the run times of all algorithms on the instance. The run time performance profile then shows the ratios on the horizontal axis and, on the vertical axis, for each algorithm, the percentage of instances with a ratio that is greater than or equal to the ratio on the horizontal axis. This implies that values in the upper left-hand corner of the graph indicate the best performance.

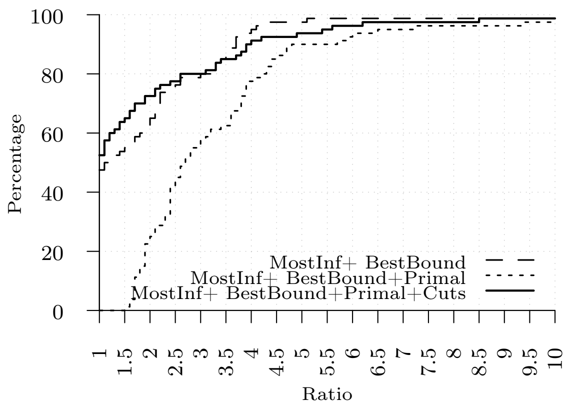
The run time performance profile of the proposed branch-and-bound algorithm for different branching settings is shown in Figure 2.4a. The node selection strategy is set to the best-bound search strategy in this figure. It is evident that the most infeasible branching strategy performs the best for the set I of instances, and it outperforms the reliability branching by a factor of up to 10. So, in the remaining, we set the branching strategy to the most infeasible unless pseudo-costs are required; In that case, we set the branching strategy to pseudo-costs branching initialized with the most infeasible branching because based on Figure 2.4a, it has the second best performance. The run time performance profile of the proposed branch-and-bound algorithm for different node selecting settings is shown in Figure 2.4b. Observe that the best-bound search strategy performs the best, and it outperforms the depth-first search strategy by a factor of up to 6.2. So, in the remaining, we set the node selecting strategy to the best-bound search strategy.



(a) Branching strategies



(b) Node selecting strategies



(c) Enhancements

Figure 2.4: The run time performance profile of the proposed branch-and-bound algorithm for different settings on mixed binary instances

The run time performance profile of the proposed branch-and-bound algorithm for different enhancements including producing an initial primal bound and producing cuts are shown in Figure 2.4c. Observe that the performance of the algorithm with no enhancement seems to be similar to the performance of the algorithm when both generating cuts and providing an initial primal bound are used. Consequently, there are two best settings for the proposed branch-and-bound algorithm. It is worth noting that the preprocessing operation is a time-consuming operation. So, we observe that if we only use the preprocessing for providing an initial primal bound then the performance of the algorithm decreases.

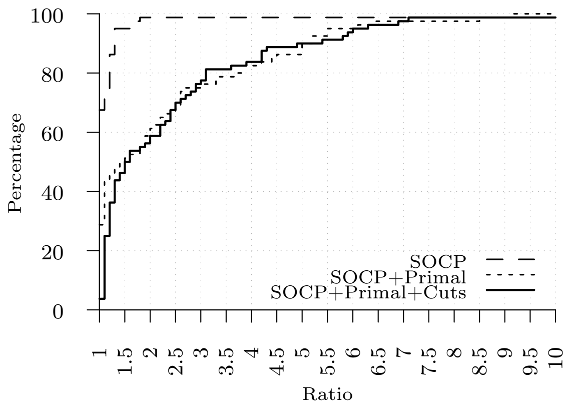


Figure 2.5: The run time performance profile of CPLEX-MI-SOCP for different settings on mixed binary instances

The run time performance profile of CPLEX-MI-SOCP for different enhancements including producing an initial primal bound and producing cuts are shown in Figure 2.5. We observe that the performance of CPLEX-MI-SOCP with no enhancement seems to be far better than the case where the cuts and/or an initial primal bound are provided to CPLEX-MI-SOCP. So, the best setting for CPLEX-MI-SOCP is to make the preprocessing inactive.

A detailed comparison between the performance of the best obtained settings for the proposed branch-and-bound algorithm and the CPLEX-MI-SOCP are shown Table 2.1 where '#N' is the number of nodes, '#LPs' is the number of (single-objective) linear programs solved, 'T(sec.)' shows the solution time in seconds, and finally '%G' shows the optimality gap percentage. Note that in this table averages over 5 instances are reported. Note too that, as discussed earlier, there are two best settings for the proposed branch-and-bound algorithm. So, in Table 2.1, 'B&B' is used for the case with no preprocessing and 'B&B+ Preprocessing' is used for the case that providing cuts and an initial primal bound is active.

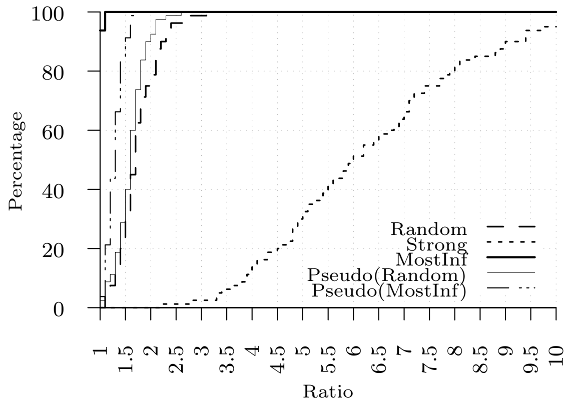
From Table 2.1, we observe that for small-sized instances, i.e., the classes with 200 and 400 constraints, CPLEX-MI-SOCP seems to perform the best on average. However, as the number of constraints increases, the performance of CPLEX-MI-SOCP decreases dramatically. For classes with 800 and 1,600 constraints, the solution time of the B&B+Preprocessing is better by a factor of around 2 on average. It is worth mentioning that both B&B and CPLEX-MI-SOCP fail to solve some of the instances with 2,400 variables and all of the instances with 3,600 variables to optimality within the imposed time limit, i.e., 7,200 seconds. In the table, when an algorithm is not able to even find a feasible solution for some of the instances of a subclass, no optimality gap is reported. Overall, the only algorithm that is able to solve all of instances to optimality is B&B+preprocessing.

Table 2.1: Performance comparison between the best settings of the branch-and-bound algorithm and CPLEX-MI-SOCP on the mixed binary instances

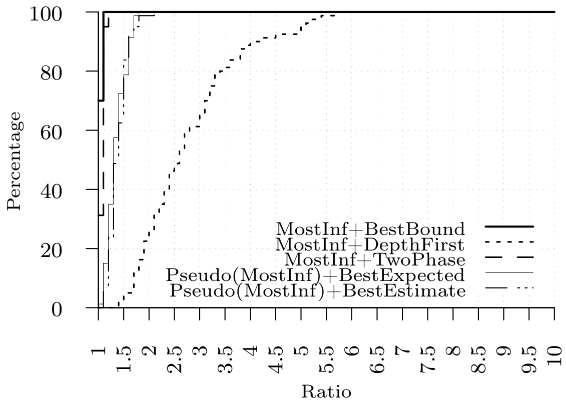
m × n	**B&B**				**B&B+Preprocessing**				**CPLEX-MI- SOCP**		
	#N	**#LPs**	**T(sec.)**	**%G**	**#N**	**LPs**	**T(sec.)**	**%G**	**#N**	**T(sec.)**	**%G**
200 × 100	31.8	1,019.2	2.2	0	5.0	272.2	10.3	0	30.0	2.7	0
200 × 200	28.6	884.4	4.1	0	8.6	370.6	21.0	0	27.0	4.5	0
200 × 300	107.0	3,180.4	24.4	0	40.6	1,460.6	37.3	0	86.2	13.9	0
200 × 400	59.4	1,704.8	20.1	0	52.2	1,637.0	51.1	0	50.6	13.9	0
Avg	**56.7**	**1,697.2**	**12.7**	**0.0**	**26.6**	**935.1**	**29.9**	**0.0**	**48.4**	**8.8**	**0.0**
400 × 200	75.8	2,137.2	20.5	0	67.0	1,861.8	32.5	0	47.4	18.9	0
400 × 400	70.6	1,943.6	45.9	0	73.4	2,455.0	72.0	0	57.8	36.1	0
400 × 600	66.2	1,826.4	85.5	0	50.6	1,597.8	92.0	0	63.8	61.8	0
400 × 800	81.4	2,208.8	174.7	0	70.6	2,151.0	140	0	73.8	93.6	0
Avg	**73.5**	**2,029.0**	**81.6**	**0.0**	**65.4**	**2,016.4**	**84.1**	**0.0**	**60.7**	**52.6**	**0.0**
800 × 400	64.2	1,764.4	90.1	0	36.6	1,133.0	70.0	0	62.6	137.4	0
800 × 800	101.8	2,493.6	457.7	0	105.8	2,993.8	258.0	0	104.4	473.5	0
800 × 1200	81.4	2,153.6	898.5	0	83.4	2,329.8	358.8	0	100.4	792.2	0
800 × 1600	76.2	1,983.6	1,496.9	0	79.4	2,328.2	488.7	0	96.4	1,068.6	0
Avg	**80.9**	**2,098.8**	**735.8**	**0.0**	**76.3**	**2,196.2**	**293.9**	**0.0**	**90.95**	**617.9**	**0.0**
1600 × 800	65.8	1,652.8	698.2	0	67.0	2,060.2	373.6	0	62.2	1,118.1	0
1600 × 1600	64.2	1,602.0	2,875.0	0	56.2	1,503.4	681.9	0	86.6	3,189.2	0
1600 × 2400	64.2	1,758.8	6,793.2	-	85.0	2,571.8	1,851.6	0	113.6	6,903.8	0.04
1600 × 3200	38.2	976.4	7,200.0	-	93.4	2,784.6	2,944.1	0	86.0	7,200.0	0.09
Avg	**58.1**	**1,497.5**	**4,391.7**	**-**	**75.4**	**2,230.0**	**1,462.8**	**0.0**	**87.1**	**4,602.9**	**0.0**

2.6.2 A Performance Comparison on Instance of Set II: Mixed General Integer Instances

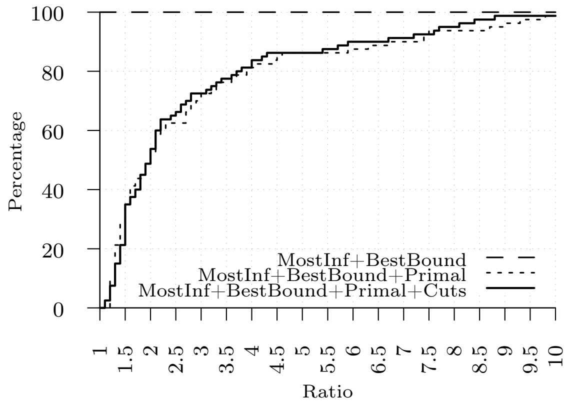
Similar to the previous section, in this section, we first compare the performance of the proposed branch-and-bound algorithm under different settings obtained by employing the proposed branching strategies, node selecting strategies, and the preprocessing technique on the instances of set II, i.e., those with no binary variables. After computing a good setting for the proposed algorithm, we then try to find a good setting for the CPLEX-MI-SOCP by exploring whether the proposed preprocessing technique is useful or not. Finally, we compare the performance of both solution methods under their best obtained settings.



(a) Branching strategies



(b) Node selecting strategies



(c) Enhancements

Figure 2.6: The run time performance profile of the proposed branch-and-bound algorithm for different settings on mixed integer instances

The run time performance profile of the proposed branch-and-bound algorithm for different branching settings is shown in Figure 2.6a. We note that, similar to the set I of instances, we have implemented the best-bound node selection strategy to compare different branching strategies. Again, it is evident that the most infeasible branching strategy performs the best for the set II of instances. So, in the remaining, we set the branching strategy to the most infeasible unless pseudo-costs are required; In that case, we set the branching strategy to pseudo-costs branching initialized with the most infeasible branching because based on Figure 2.6a, it has the second best performance. The run time performance profile of the proposed branch-and-bound algorithm for different node selecting settings is shown in Figure 2.6b. Based on Figure 2.6b, in the remaining, we set the node selecting strategy to the best-bound search strategy because it performs the best. The run time performance profile of the proposed branch-and-bound algorithm for different enhancements including producing an initial primal bound and cuts are shown in Figure 2.6c. Observe that the performance of the algorithm with no enhancement performs the best. So, in the remaining, the preprocessing is inactive for the proposed branch-and-bound algorithm.

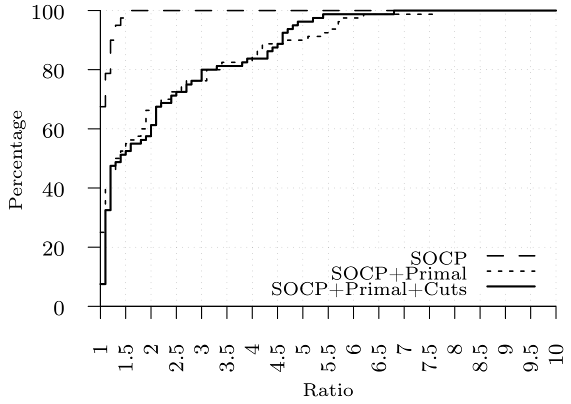


Figure 2.7: The run time performance profile of CPLEX-MI-SOCP for different settings on mixed integer instances

The run time performance profile of CPLEX-MI-SOCP for different enhancements including producing an initial primal bound and producing cuts are shown in Figure 2.7. We again observe that the performance of CPLEX-MI-SOCP with no enhancement seems to be far better than the case where the cuts and/or an initial primal bound are provided to CPLEX-MI-SOCP. So, the best setting for CPLEX-MI-SOCP is to make the preprocessing inactive.

A detailed comparison between the performance of the best obtained settings for the proposed branch-and-bound algorithm and the CPLEX-MI-SOCP are shown Table 2.2. Observe that B&B performs the best across all classes. In fact, B&B outperforms CPLEX-MI-SOCP by a factor of around 2 for all instances on average. Also, we again observe that CPLEX-MI-SOCP fails to solve some of the instances with 2,400 variables and all of the instances with 3,600 variables to optimality within the imposed time limit, i.e., 7,200 seconds.

Table 2.2: Performance comparison between the best settings of the branch-and-bound algorithm and CPLEX-MI-SOCP on the mixed integer instances

m × n	**B&B**				**CPLEX-MI-SCOP**		
	#N	**#LPs**	**T(sec.)**	**%G**	**#N**	**T(sec.)**	**%G**
200 × 100	39.0	1,169.2	2.1	0	28.2	2.3	0
200 × 200	52.2	1,586.8	5.3	0	41.4	6.4	0
200 × 300	38.2	1,165.2	6.0	0	34.0	8.2	0
200 × 400	77.4	2,190.0	14.7	0	68.0	17.0	0
Avg	**51.7**	**1,527.8**	**7.0**	**0.0**	**42.9**	**8.5**	**0.0**
400 × 200	46.6	1,457.2	10.1	0	40.4	17.9	0
400 × 400	74.6	2,101.2	28.7	0	59.8	38.9	0
400 × 600	90.6	2,601.6	54.4	0	85.6	88.5	0
400 × 800	115.4	3,252.0	93.7	0	119.2	157.0	0
Avg	**81.8**	**2,353.0**	**46.7**	**0.0**	**76.2**	**75.6**	**0.0**
800 × 400	96.6	2,592.4	74.4	0	77.6	185.8	0
800 × 800	89.4	2,498.0	154.3	0	110.4	524.0	0
800 × 1200	73.4	2,029.6	206.7	0	94.6	680.5	0
800 × 1600	86.2	2,155.2	315.9	0	95.8	914.3	0
Avg	**86.4**	**2,318.8**	**187.8**	**0.0**	**94.6**	**576.2**	**0.0**
1600 × 800	73.4	1,956.4	285.2	0	79.6	1,410.1	0
1600 × 1600	82.2	2,144.0	772.5	0	94.6	3,493.7	0
1600 × 2400	95.4	2,388.8	1,491.5	0	95.8	6,202.1	0.02
1600 × 3200	104.2	2,484.8	2,223.0	0	75.2	7,200.0	0.11
Avg	**88.8**	**2,243.5**	**1,193.6**	**0.0**	**86.3**	**4,576.7**	**0.0**

Chapter 3: A Criterion Space Search Algorithm for Mixed Integer Linear Maximum Multiplicative Programs: A Multi-objective Optimization Approach

The copyright permissions for reuse previously published material in this chapter can be found in Appendix A2.

The main contribution of this chapter is to develop the first *criterion space search algorithm* for solving MIL-MMPs for any $p \geqslant 2$. In the literature of multi-objective optimization, an algorithm is called a criterion space search algorithm if it solves a multi-objective optimization problem by working on the space of objective functions. Such algorithms transform a multi-objective optimization problem into a sequence of single-objective optimization problems that have to be solved. So, our proposed algorithm is a criterion space search algorithm because it attempts to solve a MIL-MMP by working on the space of $y_1(\boldsymbol{x}), \dots, y_p(\boldsymbol{x})$ and solving a set of single-objective mixed integer linear programs using CPLEX. We conduct an extensive computational study and show the following results:

- The proposed algorithm outperforms (mixed integer) CPLEX SOCP solver (for instances with $p \geqslant 2$) and the branch-and-bound algorithm proposed by [4] (for instances with $p = 2$) by a factor of more than 10 on many instances.

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

This chapter is organized as follows. In Section 3.2, a high-level description of the algorithm is provided. In Section 3.3, a detailed description of the proposed algorithm is given. In Section 3.4, we conduct an extensive computational study. Finally, in Section 7.3, some concluding remarks are explained.

3.1 Preliminaries

A MIL-MMP can be stated as follows,

$$\begin{aligned} \max\ & \prod_{i=1}^{p} y_i \\ \text{s.t.}\ & \boldsymbol{y} = D\boldsymbol{x} + \boldsymbol{d} \\ & A\boldsymbol{x} \leqslant \boldsymbol{b} \\ & \boldsymbol{x}, \boldsymbol{y} \geqslant \boldsymbol{0}, \quad \boldsymbol{x} \in \mathbb{R}^{n_c} \times \mathbb{B}^{n_b} \times \mathbb{Z}^{n_i}, \quad \boldsymbol{y} \in \mathbb{R}^{p}, \end{aligned} \tag{3.1}$$

where n_c, n_b, and n_i represent the number of continuous, binary, and integer decision variables, respectively. Also, D is a $p \times n$ matrix where $n := n_c + n_b + n_i$, $\boldsymbol{d}$ is a p-vector, A is 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 continuous $\mathcal{C}$, binary $\mathcal{B}$, and integer $\mathcal{I}$.

In this paper, we refer to the sets $\mathcal{X} := \{\boldsymbol{x} \in \mathbb{R}^{n_c} \times \mathbb{B}^{n_b} \times \mathbb{Z}^{n_i} : A\boldsymbol{x} \leqslant \boldsymbol{b},\ \boldsymbol{x} \geqslant \boldsymbol{0}\}$ and $\mathcal{Y} := \{\boldsymbol{y} \in \mathbb{R}^p : \boldsymbol{x} \in \mathcal{X},\ \boldsymbol{y} = D\boldsymbol{x} + \boldsymbol{d},\ \boldsymbol{y} \geqslant \boldsymbol{0}\}$ as *the feasible set in the decision space* and *the feasible set in the criterion space*, respectively. We assume that $\mathcal{X}$ is bounded (which implies that $\mathcal{Y}$ is compact) and the optimal objective value of the problem is strictly positive, i.e., there exists a $\boldsymbol{y} \in \mathcal{Y}$ such that $\boldsymbol{y} > \boldsymbol{0}$. We usually refer to $\boldsymbol{x} \in \mathcal{X}$ as a *feasible solution* and to $\boldsymbol{y} \in \mathcal{Y}$ as a *feasible point* ($\boldsymbol{y}$ is the image of $\boldsymbol{x}$ in the criterion space).

A MIL-MMP can be reformulated as a mixed integer SOCP. This procedure is explained in [3] based on the study of [51]. In this section, we first briefly review how this reformulation can be done. Observe that by introducing a new non-negative variable, γ, and a *geometric mean* constraint, Problem (3.1) can be reformulated as follows:

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

It is evident that if $\bar{\gamma}$ is the optimal objective value of the reformulated problem, then $\bar{\gamma}^p$ is the optimal objective value of Problem (3.1). Now, in order to transform the problem into a mixed integer SOCP, the above-mentioned geometric mean constraint with p terms should be replaced with an equivalent set of geometric mean constraints with exactly two terms. In order to do so, some new auxiliary variables and constraints should be introduced based on the procedure explained in [51]. The proof of transformation procedure is beyond the scope of this paper. However, interested readers can easily check/see its correctness for $p = 2$ and $p = 3$. To apply the transformation, let k be the smallest integer value such that $2^k \geqslant p$. By introducing a set of non-negative variables and constraints, the geometric mean constraint can be replaced as follows:

$$\begin{aligned}
&\max \gamma \\
&\text{s.t. } 0 \leqslant \gamma \leqslant \sqrt{\tau_1^{k-1}\tau_2^{k-1}} \\
&\quad 0 \leqslant \tau_j^l \leqslant \sqrt{\tau_{2j-1}^{l-1}\tau_{2j}^{l-1}} && \text{for } j = 1, \ldots, 2^{k-l} \text{ and } l = 1, \ldots, k-1 \\
&\quad 0 \leqslant \tau_j^0 = y_j && \text{for } j = 1, \ldots, p \\
&\quad 0 \leqslant \tau_j^0 = \gamma && \text{for } j = p+1, \ldots, 2^k \\
&\quad \boldsymbol{y} \in \mathcal{Y}.
\end{aligned}$$

The above formulation is a mixed integer SOCP since any constraint of the form $\{u, v, w \geqslant 0 : u \leqslant \sqrt{vw}\}$ is equivalent to $\{u, v, w \geqslant 0 : \sqrt{u^2 + (\frac{v-w}{2})^2} \leqslant \frac{v+w}{2}\}$. Now, one can use a commercial mixed integer second-order cone programming solver such as CPLEX to solve any MIL-MMP. However, the goal of this paper is to develop an effective and different solution approach based on multi-objective optimization. Hence, we provide the following critical definition and proposition adapted from [4]. Note that a similar definition and proposition can also be found in [53] and [54, 55].

Definition 3.1. A feasible solution $\boldsymbol{x} \in \mathcal{X}$ is called *efficient*, if there is no other $\boldsymbol{x}' \in \mathcal{X}$ such that

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

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 3.1. *An optimal solution of Problem* (3.1)*, 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.*

Proposition 3.1 implies that Problem (3.1) is equivalent to $\max_{\boldsymbol{y} \in \mathcal{Y}_N} \prod_{i=1}^{p} y_i$. Therefore, this problem is precisely optimization over the efficient set. Hence, instead of searching the entire feasible set, we propose an algorithm which looks for an optimal solution by searching over the set of nondominated points.

As we will explain in Section 3.3.1, the proposed algorithm sometimes adds the so-called *no-good* constraints to the problem to exclude a previously found solution from the future search region [56]. It is known that no-good constraints for problems with general integer decision variables are naturally non-linear [57]. Although it is possible to linearize such constraints, it is more convenient to add no-good constraints to problems involving only binary decision variables because they will be naturally linear. So, in the rest of this study, whenever we want to solve a MIL-MMP by the proposed algorithm, we assume that all integer variables are transformed into binary decision variables using the standard binary transformation procedure. Specifically, because we have assumed that $\mathcal{X}$ is bounded then for any integer decision variable x a global upper bound should be computable, i.e., $x \leqslant u$. Hence, the variable x can be replaced by introducing $v := \lfloor log_2 u \rfloor + 1$ new binary variables as follows,

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

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

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

3.2 High-level Description

In this section, we provide a high-level description of our algorithm. In each iteration, the algorithm starts by computing a *locally* nondominated point, denoted by $\bar{\boldsymbol{y}}$. Note that the term "locally" is used because in each iteration the algorithm removes a set of feasible solutions from the problem by adding some constraints. So, the point $\bar{\boldsymbol{y}}$ will be certainly a nondominated point for the restricted feasible set but not necessarily for the entire feasible set. An illustration of the nondominated frontier of Problem (3.1) when $p = 2$ and also $\bar{\boldsymbol{y}}$ (obtained in the first iteration) can be found in Figure 3.1a. As an aside, note that the nondominated frontier of a multi-objective mixed integer linear program can be a very complicated shape. For example, when $p = 2$, it may contain some continuous segments, points, open or half-open line segments [58].

Considering that $\bar{\boldsymbol{y}}$ is a feasible point, $\prod_{i=1}^{p} \bar{y}_i$ provides a global lower bound for Problem (3.1). However, a better lower bound can be (possibly) computable based on a solution corresponding to $\bar{\boldsymbol{y}}$, denoted by $\bar{\boldsymbol{x}}$. In order to find a better lower bound, we fix the value of the binary variables in Problem (3.1) to their corresponding values in solution

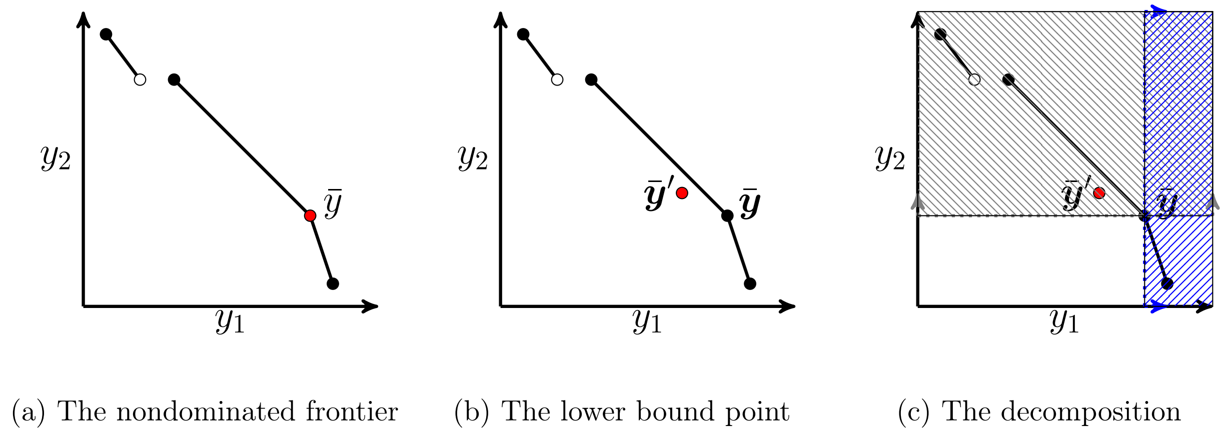


Figure 3.1: A high-level description of the proposed algorithm when $p = 2$

$\bar{\boldsymbol{x}}$, and by so doing, the resulted problem will be a L-MMP. So, one can optimize such a problem to obtain a (possibly) better feasible point in the criterion space, denoted by $\bar{\boldsymbol{y}}'$, where $\prod_{i=1}^{p} \bar{y}_i \leqslant \prod_{i=1}^{p} \bar{y}'_i$. An illustration of $\bar{\boldsymbol{y}}'$ can be found in Figure 3.1b when $p = 2$.

Note that $\bar{\boldsymbol{y}}'$ is not necessarily a nondominated point for the original problem. However, by construction, it will provide the best possible lower bound when binary decision variables are fixed to their corresponding values in $\bar{\boldsymbol{x}}$. So, the key idea of the algorithm is to add the so-called no-good constraint to the formulation for ensuring that the *binary support vector*, i.e., the vector of the values of binary decision variables, associated with $\bar{\boldsymbol{x}}$ will be excluded from the search (in the future iterations). Another key component of the algorithm is that it will decompose the criterion space in each iteration based on $\bar{\boldsymbol{y}}$. Since $\bar{\boldsymbol{y}}$ is a nondominated point for the restricted problem, one can immediately remove the dominated part of the criterion space associated with $\bar{\boldsymbol{y}}$ and continue the search in the remaining areas. The new areas will be p new boxes, an illustration of which are shown in Figure 3.1c when $p = 2$, i.e., the box with $y_1 \geqslant \bar{y}_1$ and the box with $y_2 \geqslant \bar{y}_2$.

It is evident that one can just simply repeat the process explained above in new boxes to be able to find an optimal solution of Problem (3.1). However, in order to speed up the searching process, we employ a procedure to calculate a global upper/dual bound for the problem. In other words, in each iteration, when exploring a box, we first compute an upper bound. If the upper bound is not strictly better than the best lower/primal bound found, then there is no need to search the box anymore. In order to obtain an upper bound, we can simply drop the integrality conditions in Problem (3.1). Note that, since our problem is in maximization form, we use the terms “primal bound" and “lower bound", and similarly the terms “dual bound" and “upper bound" interchangeably in this research.

3.3 Detailed Description

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

3.3.1 No-good Constraints

As mentioned in Section 3.2, no-good constraints, also known as *tabu* constraints [56, 59, 60], play an important role in the proposed algorithm by excluding the studied feasible binary support vectors from the future searches. In the proposed algorithm, we maintain a list of the so-called tabu feasible solutions, denoted by TABU. Note that we assume that all general integer decision variables of Problem (3.1) are transformed to binary decision variables. Therefore, solution $\boldsymbol{x}' \in$ TABU consists of two support vectors: continuous and binary. For each $\boldsymbol{x}' \in$ TABU, the algorithm will add the following no-good constraint to ensure that the binary support vector associated with $\boldsymbol{x}'$ will be excluded from the search,

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

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

3.3.2 Computing Primal Bound

By Proposition 3.1, any efficient solution is expected to provide a good (global) lower/primal bound for Problem (3.1). Therefore, the algorithm attempts to compute a *locally* efficient solution in each iteration. In multi-objective optimization, it is well-known (see for instance [49]) that optimizing any positive weighted summation of the objective functions over the feasible set must return an efficient solution (if any exists). So, in this research, we simply assume that the weights are equal to one, and therefore, we optimize/maximize the corresponding objective function, i.e., $\sum_{i=1}^{p} y_i$, over the feasible set, i.e., $y \in \mathcal{Y}$, but with some additional constraints.

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

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

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

where TABU is the list of tabu solutions whose corresponding binary support vectors should be excluded from the search. Also, $\boldsymbol{l} \in \mathbb{R}^p_+$ is the vector of lower bound values for objective functions $y_1, \ldots, y_p$. Note that as explained in Section 3.2, in each iteration, the algorithm needs to search a box defined by $\{\boldsymbol{y} \in \mathbb{R}^p : \boldsymbol{l} \leqslant \boldsymbol{y} \leqslant +\infty\}$. So, the vector $\boldsymbol{l}$ is basically restricting the search to a box in the above optimization problem.

After calling WSO($\boldsymbol{l}$,TABU) and computing $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$, if it is feasible, then the algorithm will fix the value of binary decision variables by setting them equal to values of the binary support vector of $\bar{\boldsymbol{x}}$ in Problem (3.1),

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

Note that in this continuous optimization problem there exist no additional constraints (other than fixing constraints). This is important because it guarantees that by solving this continuous optimization problem we can compute the best possible solution for a fixed binary support vector. In order to solve this continuous optimization problem, we first transform it to a continuous SOCP (using the same procedure explained in Section 3.1) and then solve it by a suitable solver, e.g., CPLEX. We denote the operation of computing $(\bar{\boldsymbol{x}}', \bar{\boldsymbol{y}}')$ by SOCP-LB($\bar{\boldsymbol{x}}$).

3.3.3 Computing Dual Bound

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

Observation 3.1. Let $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ be an optimal solution of WSO($\boldsymbol{l}$,TABU) at a particular iteration. Due to optimality, the following cut can be used when searching the box associated with $\boldsymbol{l}$ in the subsequent iterations,

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

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

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

where $\mathcal{Y}_R$ is a relaxation of $\mathcal{Y}$ obtained by dropping the integrality condition on binary decision variables, i.e., $x_i \in [0,1]$ for $i \in \mathcal{B}$. Note that, as explained in Section 3.2, the current box (in each iteration) is obtained as a result of decomposing a larger box, referred to as *Parent* box, in one of the previous iterations. By this explanation, $\bar{y}_i^{Parent}$ is the feasible point found by solving the weighted sum optimization problem for the parent box. So, $\sum_{i=1}^p y_i \leqslant \sum_{i=1}^p \bar{y}_i^{Parent}$ is the cut derived based on Observation 3.1.

In order to solve this continuous optimization problem, we again first transform it to a continuous SOCP solver (using the same procedure explained in Section 3.1) and then solve it by a suitable solver, e.g., CPLEX. We denote the operation of computing $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$ by SOCP-UB$(\boldsymbol{l}, \bar{\boldsymbol{y}}^{Parent}, \textsc{Tabu})$.

```
Input: A feasible instance of Problem (3.1)
$Queue.create(\textsc{Tree})$
$List.create(\textsc{Tabu})$
$\boldsymbol{l} \leftarrow -\infty$; $\bar{\boldsymbol{y}}^{Parent} \leftarrow +\infty$
$\textsc{G}_{\textsc{LB}} \leftarrow -\infty$; $\textsc{G}_{\textsc{UB}} \leftarrow +\infty$
$\textsc{Tree}.add\big((\boldsymbol{l}, \bar{\boldsymbol{y}}^{Parent})\big)$
while not $Queue.empty(\textsc{Tree})$ & $\textsc{G}_{\textsc{UB}} - \textsc{G}_{\textsc{LB}} \geqslant \varepsilon_1$ & $\frac{\textsc{G}_{\textsc{UB}} - \textsc{G}_{\textsc{LB}}}{\textsc{G}_{\textsc{UB}}} \geqslant \varepsilon_2$ do
    $\textsc{Tree}.PopOut\big((\boldsymbol{l}, \bar{\boldsymbol{y}}^{Parent})\big)$
    $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) \leftarrow$ SOCP-UB$(\boldsymbol{l}, \bar{\boldsymbol{y}}^{Parent}, \textsc{Tabu})$
    Update $\textsc{G}_{\textsc{UB}}$
    if $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) \neq$ (null,null) & $\prod_{i=1}^p \tilde{y}_i > \textsc{G}_{\textsc{LB}}$ then
        $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \leftarrow$ WSO$(\boldsymbol{l}, \textsc{Tabu})$
        if $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \neq$ (null,null) then
            $(\bar{\boldsymbol{x}}', \bar{\boldsymbol{y}}') \leftarrow$ SOCP-LB$(\bar{\boldsymbol{x}})$
            if $\prod_{i=1}^p \bar{y}_i' > \textsc{G}_{\textsc{LB}}$ then
                $\textsc{G}_{\textsc{LB}} \leftarrow \prod_{i=1}^p \bar{y}_i'$
                $(\boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow (\bar{\boldsymbol{x}}', \bar{\boldsymbol{y}}')$
            end
            $\textsc{Tabu}.add(\bar{\boldsymbol{x}})$
            foreach $i \in \{1, 2, .., p\}$ do
                $\boldsymbol{l}^i \leftarrow \boldsymbol{l}$
                $l_i^i \leftarrow \bar{y}_i$
                $\textsc{Tree}.add(\boldsymbol{l}^i, \bar{\boldsymbol{y}})$
            end
        end
    end
end
return $(\boldsymbol{x}^*, \boldsymbol{y}^*)$
```

Algorithm 3: The proposed algorithm

3.3.4 The Proposed Algorithm

In Sections 3.3.1-3.3.3, we explained the key components of the algorithm. So, we can now present a precise description of the proposed algorithm which can also be found in

Algorithm 3. The algorithm maintains a queue of nodes, denoted by TREE. Each node in the queue contains two vectors denoted by $\boldsymbol{l}$ and $\bar{\boldsymbol{y}}^{Parent}$. As mentioned before, $\boldsymbol{l}$ is a lower bound point in the criterion space that defines a box. Also, $\bar{\boldsymbol{y}}^{Parent}$ is the feasible point obtained by solving the weighted sum optimization problem for the parent of each node. We initialize the queue (of nodes) by $\boldsymbol{l} = -\infty$ and $\bar{\boldsymbol{y}}^{Parent} = +\infty$. The algorithm also maintains three other pieces of information including a tabu list which is denoted by TABU, a global upper/dual bound denoted by $\mathrm{G_{UB}}$, and a global lower/primal bound denoted by $\mathrm{G_{LB}}$. At the beginning, TABU is empty and we set $\mathrm{G_{UB}} = +\infty$ and $\mathrm{G_{LB}} = -\infty$. The algorithm explores the queue as long as it is nonempty and $\mathrm{G_{UB}} - \mathrm{G_{LB}} \geqslant \varepsilon_1$ and $\frac{\mathrm{G_{UB}} - \mathrm{G_{LB}}}{\mathrm{G_{UB}}} \geqslant \varepsilon_2$, where $\varepsilon_1, \varepsilon_2 \in (0,1)$ are the user-defined absolute and relative optimality gap tolerances, respectively. At the end, the algorithm returns the best solution found and its image in the criterion space, denoted by $(\boldsymbol{x}^*, \boldsymbol{y}^*)$. Next, we explain how each node of the queue is explored.

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

The algorithm first calls WSO($\boldsymbol{l}$, TABU) to compute $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$. If $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) = (null, null)$, i.e., no feasible solution exists, then no further exploration is required and so a new iteration should be started (if possible). Otherwise, the algorithm calls SOCP-LB($\bar{\boldsymbol{x}}$) to compute $(\bar{\boldsymbol{x}}', \bar{\boldsymbol{y}}')$, i.e., the best possible primal bound for the binary support vector associated with $\bar{\boldsymbol{x}}$. If the obtained solution is better than the best global solution then the algorithm updates

G_{LB} and $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ accordingly. Next, $\bar{\boldsymbol{x}}$ will be added to the tabu list, and p new nodes will be created. The difference between the new nodes are simply their associated boxes. Specifically, for each $i \in \{1, \ldots, p\}$, the node $(\boldsymbol{l}^i, \bar{\boldsymbol{y}})$ will be added where $\boldsymbol{l}^i$ defines the associated box such that $l^i_j = l_j$ for each $j \in \{1, \ldots, p\} \backslash \{i\}$ and $l^i_i = \bar{y}_i$.

Observation 3.2. The proposed algorithm is finite. This observation is true due to the following two reasons: (1) In each iteration a new binary support vector will be added to the tabu list by construction; and (2) The number of binary support vectors is finite because $\mathcal{X}$ is bounded by assumption.

3.4 Computational Study

In this section, we compare the performance of our proposed algorithm (referred to as Algorithm 3) with the performance of the mixed integer SOCP solver of CPLEX (referred to as SOCP) and the algorithm proposed in our recent study [4] for instances with $p = 2$ (referred to as B&B). We implement our algorithm in C++ and use CPLEX 12.7 through this computational study. The instances and the C++ implementation of our algorithm used in this computational study can be found in https://github.com/paymanghasemi. The computational experiments are conducted on a Dell PowerEdge R360 with two Intel Xeon E5-2650 2.2 GHz 12-Core Processors (30MB), 128GB RAM, the RedHat Enterprise Linux 6.8 operating system, and using a single thread. Both the absolute and relative optimality gap tolerances, ε_1 and ε_2, are set to 10^{-6} in this computational study. Also, a time limit of 7200 seconds is imposed for solving each instance for all algorithms. Next, we explain how the instances are generated.

In our computational study, we test the performance of our algorithm on instances with $p \in \{2, 3, 4\}$ objective functions. We note that, theoretically, all solution methods discussed in the paper should handle instances with any arbitrary value of p. However, in practice, as p increases, the optimal objective value of a MIL-MMP tends to grow dramatically which directly/indirectly result in numerical instabilities in all existing solution methods/solvers. Therefore, to avoid numerical instabilities, we did not generate instances with $p > 4$ in this computational study. With this in mind, for any $p \in \{2, 3, 4\}$, we generate four classes of instances including pure integer instances ($n_i = n$), pure binary instances ($n_b = n$), mixed integer instances ($n_i = 0.5n$ and $n_c = 0.5n$), and mixed binary instance ($n_b = 0.5n$ and $n_c = 0.5n$). Each class contains 16 subclasses of instances based on the dimensions of the matrix $A_{m\times n}$, and each subclass contains 10 instances. So, each class has a total of 160 instances and the total number of instances used in this computational study is $3 \times 4 \times 160 = 1920$.

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

As mentioned in Section 3.1, each integer decision variable should be converted to (a set of) binary decision variables when using our proposed algorithm, i.e., Algorithm 3. Note that when solving an instance with other methods, no transformation is required. Since the maximum value for components of vector $\boldsymbol{b}$ is 150 and the minimum value for entries of matrix A is 10, each decision variable can get a maximum value of 15. Therefore, in the process of converting integer variables to binary variables, the integer variable x_i is converted to four binary variables as follows,

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

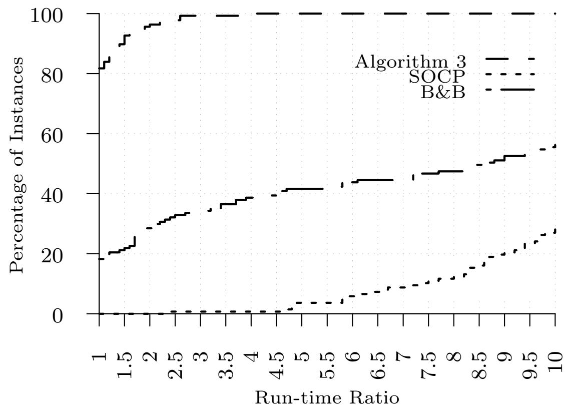
$$z_{i_1} + 2z_{i_2} + 4z_{i_3} + 8z_{i_4} \leqslant 15.$$

Note that the additional constraints are redundant and there is no need to add them to the model. Hence, for pure integer instances studied in this paper, the number of decision variables increases by a factor of 4 when using Algorithm 3.

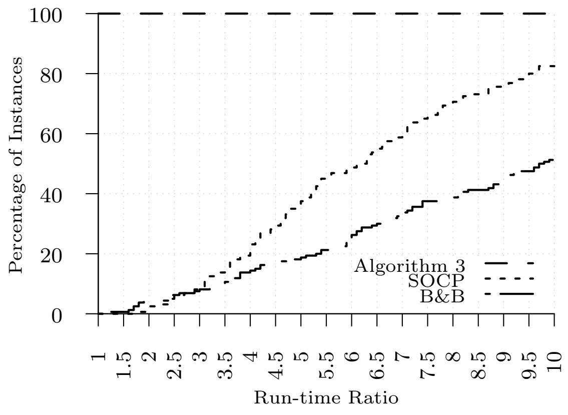
It is worth mentioning that in this computational study, we frequently use *performance profiles* [52] to compare different algorithms in terms of their solution times. A performance profile presents cumulative distribution functions for a set of algorithms being compared with respect to a specific performance metric. The run-time performance profile for a set of algorithms is constructed by computing for each algorithm and for each instance the ratio of the run-time of the algorithm on the instance and the minimum of the run-times of all algorithms on the instance. The run-time performance profile then shows the ratios on the horizontal axis and, on the vertical axis, for each algorithm, the percentage of instances with a ratio that is smaller than or equal to the ratio on the horizontal axis. This implies that values in the upper left-hand corner of the graph indicate the best performance.

3.4.1 Two Objectives ($p = 2$)

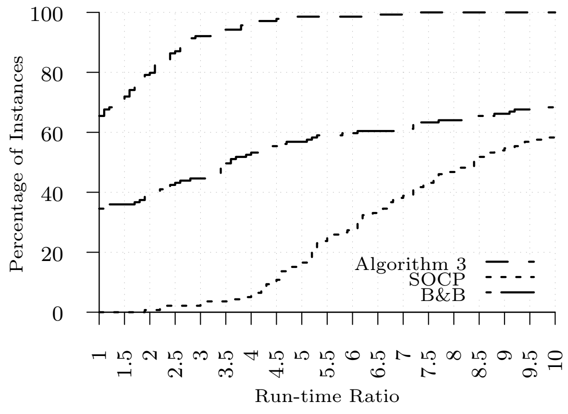
In this section, we compare the performance of Algorithm 3, SOCP, and B&B. Note that because B&B is developed for instances with $p = 2$, we use B&B only in this section. Figure 3.2 shows the run-time performance profiles of all three approaches for different classes of instances when $p = 2$. From Figure 3.2, we observe that our proposed algorithm significantly outperforms the other algorithms. For example, for pure binary instances, Algorithm 3 outperforms SOCP and B&B by a factor of at least 10 on more than 45% and 75% of instances, respectively. Observe that, for instances involving general integer decision variables, the performance of Algorithm 3 decreases because of the need to transform integer variables to binary variables which increases the size of the instance. However, Algorithm 3 is still significantly better than other approaches. For example, for mixed integer instances, Algorithm 3 outperforms B&B and SOCP by a factor of at least 2.5 on more than 40% and 60% of instances, respectively.



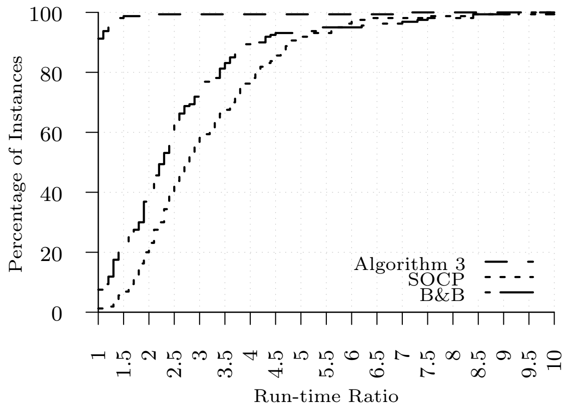
(a) Pure binary



(b) Mixed binary



(c) Pure integer



(d) Mixed integer

Figure 3.2: Performance profiles for instances with $p = 2$

Tables 3.1-3.2 provide a detailed comparison between SOCP and Algorithm 3. For the detailed comparison between the B&B and Algorithm 3, interested readers may refer to Appendix B1. In these tables, '#N' is the number of nodes explored, 'T(sec.)' shows the solution time in seconds, '#S' is the number of instances for which at least a feasible solution (other than zero) is found, and finally, '%G' shows the (relative) optimality gap percentage. Note that, in the tables, the numbers are the averages over 10 instances. In cases that an algorithm was not able to find a feasible solution for all 10 instances (in a subclass), the numbers are the averages over #S. If column #S is not reported for a method then the method was able to find at least one feasible solution (other than zero) for all instances.

Similarly, if column %G is not reported for a method then the method was able to solve all instances to optimality.

Table 3.1: Performance comparison between Algorithm 3 and SOCP on pure binary and mixed binary instances with $p = 2$

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

Table 3.2: Performance comparison between Algorithm 3 and SOCP on pure integer and mixed integer instances with $p = 2$

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

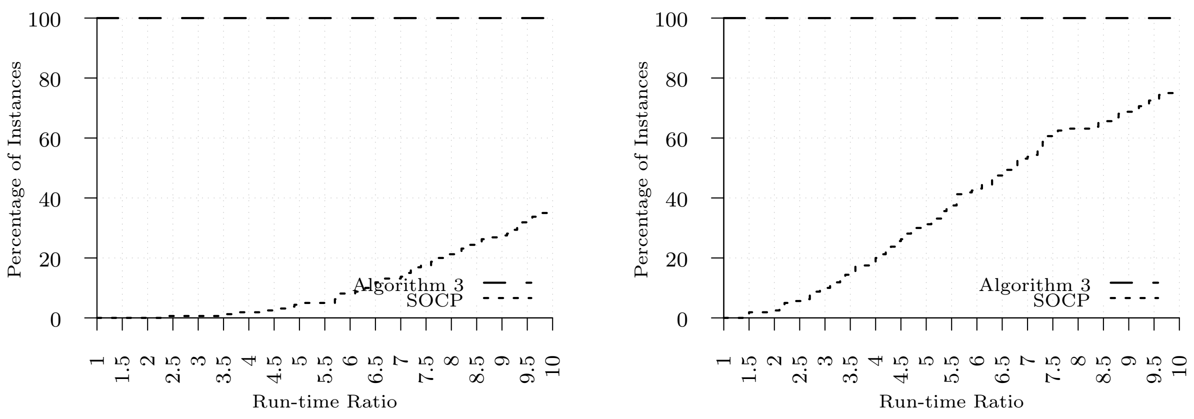
From Tables 3.1-3.2, we observe that, in pure binary and pure integer instances, there are some cases in which SOCP fails to find even a feasible solution (other than zero).

According to [61], there is a parameter to change the convergence tolerance. Changing this tolerance to a smaller value may result in greater numerical precision of the solution, but also increases the chance of a convergence failure in the algorithm and consequently may result in no solution at all. According to the manual, the smallest value for this tolerance is 10^{-12} and its proposed value is 10^{-7}. In order to overcome the convergence problem, we examined different values for this parameter, i.e., $\{10^{-6}, 10^{-7}, 10^{-8}, \cdots, 10^{-12}\}$, during the course of this study. However, we employed 10^{-6} and 10^{-12} in this paper because these values performed the best. Specifically, in this computational study, the default value is 10^{-12}. However, SOCP fails when solving pure binary and pure integer instances. Consequently, for those instances, we also provide results when the parameter is set to 10^{-6}. Note that, since SOCP fails (especially when the convergence parameter is 10^{-12}) to solve some of the instances, for any performance profile in this computational study, the graphs are drawn only based on the instances that are solved by all algorithms.

In light of the above, Figure 3.3 and Table 3.3 show the results on pure binary and pure integer instances when the parameter is set to 10^{-6}. From Figure 3.3, we observe that our proposed algorithm solves almost 65% of the pure binary instances and almost 25% of the pure integer instances 10 times faster than SOCP. Also, Table 3.3 shows that SOCP was able to find a feasible solution (other than zero) for all pure binary and pure integer instances when the parameter is set to 10^{-6}.

3.4.2 Three Objectives ($p = 3$)

The run-time performance profiles of Algorithm 3 and SOCP for pure binary and mixed binary instances with $p = 3$ are shown in Figures 3.4a and 3.4b, respectively, and the table containing the detailed comparison can be found in Appendix B2. We observe from Figure 3.4a that our proposed algorithm has solved almost 50% of the pure binary instances at least 10 times faster than SOCP. Also, it was able to solve 20% of the mixed binary instances at least 5 times faster than SOCP. We note that, among 160 instances with



(a) Pure binary

(b) Pure integer

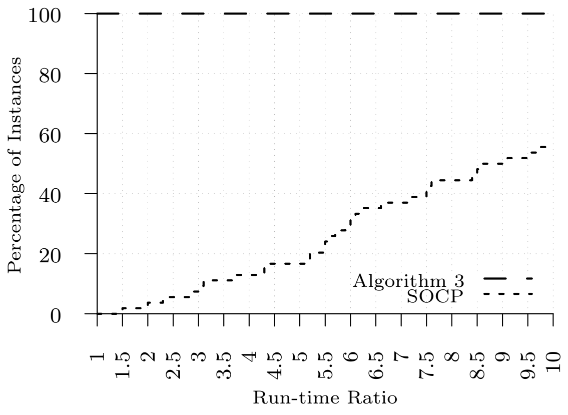
Figure 3.3: Performance profile of instances with $p = 2$ for the convergence tolerance of 10^{-6}

Table 3.3: Performance comparison between Algorithm 3 and SOCP on instances with $p = 2$ for the convergence tolerance of 10^{-6}

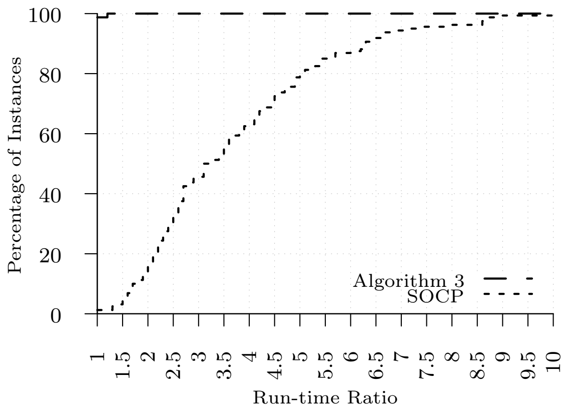
	Pure binary					Pure integer				
	Algorithm 3		SOCP			Algorithm 3		SOCP		
$m \times n$	**#N**	**T(sec.)**	**#S**	**%G**	**T(sec.)**	**#N**	**T(sec.)**	**#S**	**%G**	**T(sec.)**
200×100	6.80	1.02	10	0	6.47	7.20	2.45	10	0	6.03
200×200	6.40	2.16	10	0	19.99	7	7.42	10	0	28.12
200×300	6.20	4.90	10	0	49.93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

pure binary variables, SOCP was able to find a feasible solution (other than zero) for only 54 instances and only solved 40 (out of 54) instances to optimality.

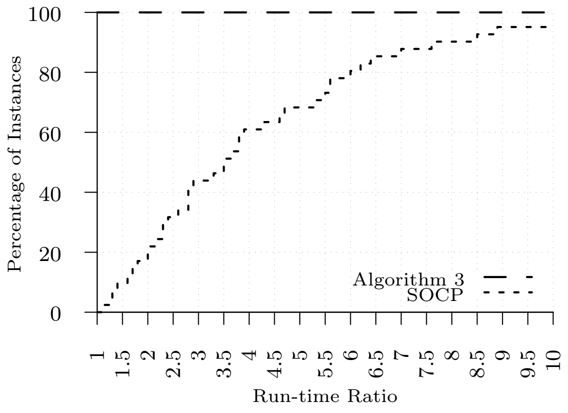
Figures 3.4c and 3.4d illustrate the run-time performance profiles for instances with pure integer and mixed integer variables, respectively, and the table containing the detailed comparison can be found in Appendix B2. Observe that although our proposed algorithm



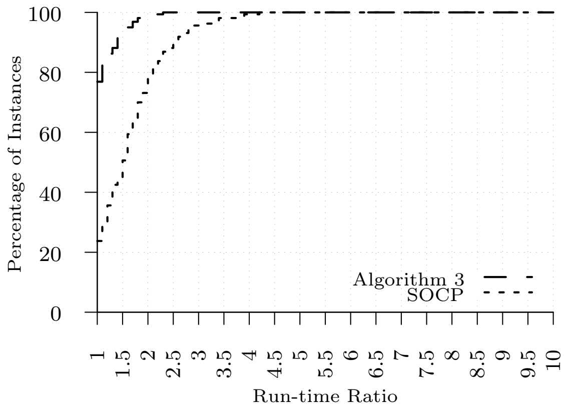
(a) Pure binary



(b) Mixed binary



(c) Pure integer

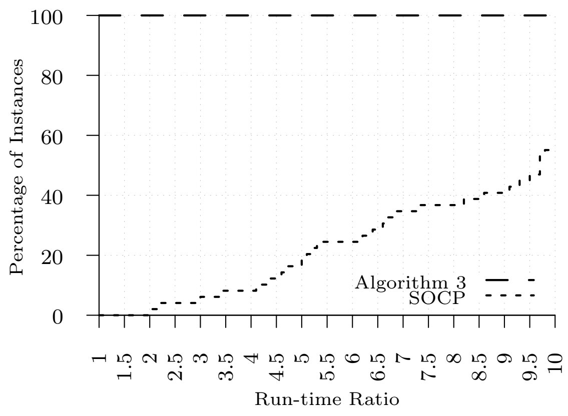


(d) Mixed integer

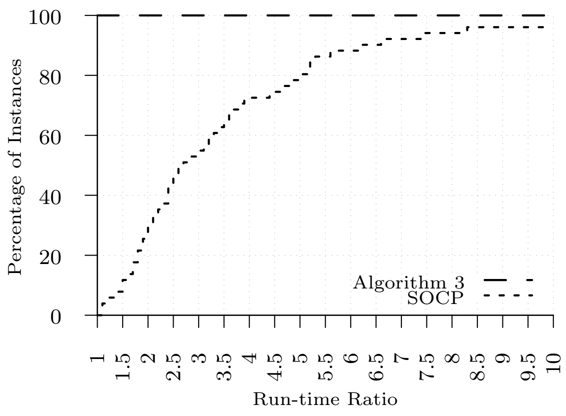
Figure 3.4: Performance profile of instances with $p = 3$

requires to solve larger-sized instances (because of the transformation), it has solved 20% of the pure integer instances and 20% of the mixed integer instances 6 times and 2 times faster than SOCP, respectively. Also, it is worth mentioning that SOCP was able to find a feasible solution (other than zero) for only 41 out of 160 instances.

Figure 3.5 shows the run-time performance profiles of the two algorithms for pure binary and pure integer instances when convergence tolerance is set to 10^{-6}, and the table containing the detailed comparison can be found in Appendix B2. Again, we observe that in both cases, our proposed algorithm outperforms SOCP. As shown in Figure 3.5a, Algorithm 3 has solved almost 45% of the pure binary instances 10 times faster than SOCP. Similarly, from Figure 3.5b, we observe that our algorithm solved 20% of the instances 5 times faster than SOCP.



(a) Pure binary

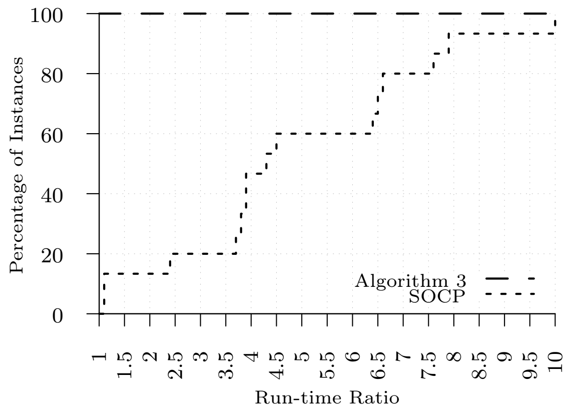


(b) Pure integer

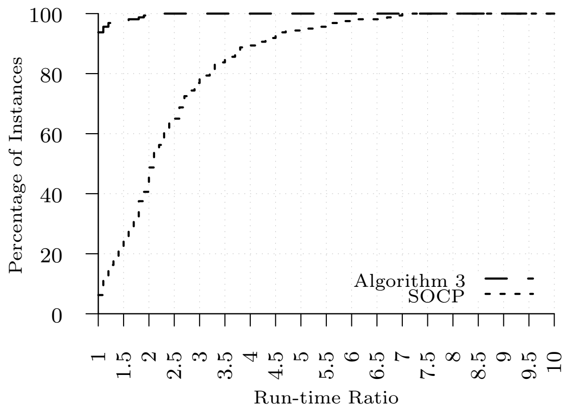
Figure 3.5: Performance profile of instances with $p = 3$ for the convergence tolerance of 10^{-6}

3.4.3 Four Objectives ($p = 4$)

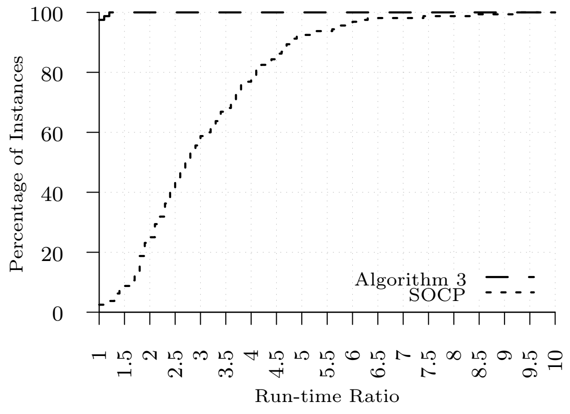
In this section, we study the performance of Algorithm 3 and SOCP for instances with $p = 4$. As an aside, we note that during the course of this study, we realized that as p increases, SOCP fails on more instances, i.e., it cannot find even a feasible solution (other than zero), when the convergence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



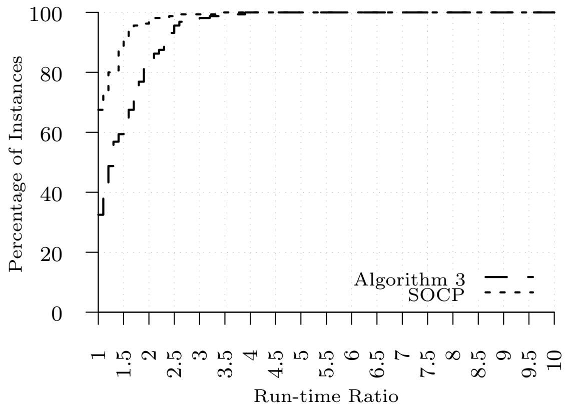
(a) Pure binary



(b) Mixed binary



(c) Pure integer



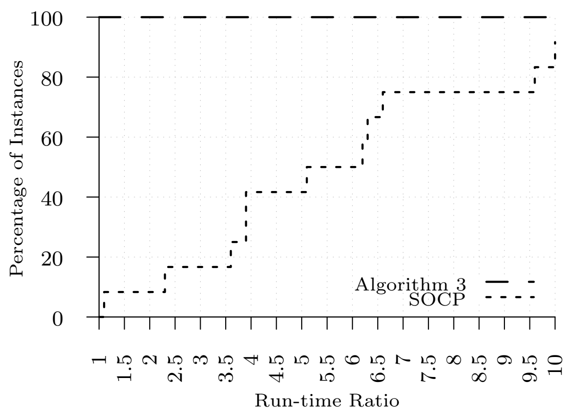
(d) Mixed integer

Figure 3.6: Performance profile of instances with $p = 4$

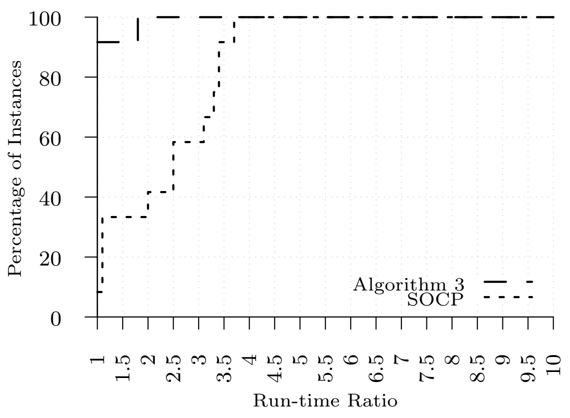
Figures 3.6a and 3.6b show the run-time performance profiles of Algorithm 3 and SOCP on instances with pure binary and mixed binary variables, respectively. The table containing the detailed comparison can be found in Appendix B3. As shown in Figure 3.6a, our proposed algorithm has solved 20% of the instances with pure binary variables at least 7.5 times faster than SOCP. Similarly, Figure 3.6b shows that our algorithm has solved 20% of the instances with mixed binary instances at least 3 times faster than SOCP.

Figures 3.6c and 3.6d provide the run-time performance profiles of the two solution methods on instances with pure integer and mixed integer variables, respectively. The table containing the detailed comparison can be found in Appendix B3. From Figure 3.6c, we observe that our proposed algorithm solves almost 20% of the instances with pure integer variables at least 4 times faster. However, from Figure 3.6d, we observe that for the instances with mixed integer variables, SOCP outperforms Algorithm 3 by solving almost 20% of the instances at least 2 times faster than our algorithm. But, there is an important point that should be considered for this observation. Based on the detailed comparison of the two algorithms, given in Appendix B3 (Table B3.2), we can observe that SOCP performs better for only small-sized instances, and as the size of the instances increases, our algorithm tends to solve instances faster. As an example, for the instances with 200 constraints, the average times are 9.43 seconds and 13.41 seconds for SOCP and Algorithm 3, respectively. However, for the instances with 800 constraints, the average times are 545.91 seconds and 478.69 seconds for SOCP and Algorithm 3, respectively.

Figures 3.7a and 3.7b show the run-time performance profiles for pure binary and pure integer instances, respectively, when the convergence tolerance parameter is set to 10^{-6} for SOCP. The table containing the detailed comparison can also be found in Appendix B3. From Figure 3.7, we observe that our algorithm solves 20% of the pure binary instances and 20% of the pure integer instances at least 9.5 and 3.5 times faster than SOCP, respectively.



(a) Pure binary



(b) Pure integer

Figure 3.7: Performance profile of instances with $p = 4$ and the convergence tolerance of 10^{-6}

3.4.4 Linearization

Note that pure binary and pure integer instances can be linearized and be solved by integer programming solvers such as CPLEX. However, the focus of this section will be only on pure binary instances with $p = 2$ as their linearization is easier and less complicated compared to other variants of MIL-MMPs, i.e. instances with integer variables or $p > 2$. As an example, consider the following objective function,

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

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 two constraints and introducing a new binary variable q,

$$\begin{gathered} q \leqslant x_i \\ q \leqslant x_j \\ x_i + x_j - 1 \leqslant q \end{gathered}$$

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

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

Following this approach, we linearized all pure binary instances with $p = 2$, and Table 3.4 provides the detailed comparison between the performance of Algorithm 3, SOCP, and linearization approach for the pure binary instances with $p = 2$. Observe that the performance of the linearization method is even worse than SOCP. In many instances, CPLEX was unable to find a feasible solution (other than zero) for linearized models. Note that, pure binary instances with $p = 2$ are in some sense the easiest class of instances for linearization. So, it is expected that the performance of the linearization technique to be even worse for other classes or larger values of p.

Table 3.4: Performance comparison between Algorithm 3, SOCP, and linearization method on pure binary instances with $p = 2$

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

Chapter 4: Exact Solution Approaches for Integer Linear Generalized Maximum Multiplicative Programs Through the Lens of Multi-objective Optimization

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 [62]. Specifically, we need to first change π_i to a positive integer number for all $i = 1, \ldots, p$ by multiplying all of them with 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 (in terms of non-linearity). 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 [63]. However, [3] 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 [51] 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 [3, 18].

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 [4, 1]. However, for MIL-GMMPs with $p = 2$, [4] proposed a faster algorithm by incorporating the linear programming based approach developed by [3] for L-GMMPs in an effective branch-and-bound framework. Recently, [1] proposed an even more effective approach for solving MIL-GMMPs with $p \geqslant 2$ 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 [1]. We show that the modified algorithm significantly outperforms the one proposed by [1] 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 [1], 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 finite number of single-objective integer linear programs to compute an optimal solution (of a nonlinear problem) 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. To the best of our knowledge, this procedure has not been employed in any existing methods.

- 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, the 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. Finally, it is worth mentioning that, as a side computational effort, we have compared the performance of our approaches on three different case studies. Interested readers may refer to Appendices 7.5 to learn about them.

The remainder of this chapter is organized as follows. In Section 4.1, we give preliminaries. In Section 4.2, we introduce a high-level description of our proposed algorithms and also their key operations. In Section 4.3, the detailed descriptions of the proposed algorithms are given. In Section 4.4, an extensive computational study is provided. Finally, in Section 7.3, some concluding remarks are provided.

4.1 Preliminaries

An IL-GMMP can be stated as

$$\begin{aligned} \max\ & \prod_{i=1}^{p} y_i^{\pi_i} \\ \text{s.t. } & \boldsymbol{y} = D\boldsymbol{x} + \boldsymbol{d} \\ & A\boldsymbol{x} \leqslant \boldsymbol{b} \\ & \boldsymbol{x}, \boldsymbol{y} \geqslant \boldsymbol{0}, \quad \boldsymbol{x} \in \mathbb{B}^{n_b} \times \mathbb{Z}^{n_i}, \quad \boldsymbol{y} \in \mathbb{R}^p, \end{aligned} \tag{4.1}$$

where n_b and n_i represent the number of binary and integer decision variables, respectively, D is an $p \times n$ matrix where $n := n_b + n_i$, $\boldsymbol{d}$ is an p-vector, A is an $m \times n$ matrix, and $\boldsymbol{b}$ is an m-vector. For notational convenience, we partition the index set of variables $\mathcal{N} := \{1, 2, ..., n\}$ into binary $\mathcal{B}$ and integer $\mathcal{I}$. In this paper, the sets $\mathcal{X} := \{\boldsymbol{x} \in \mathbb{B}^{n_b} \times \mathbb{Z}^{n_i} : A\boldsymbol{x} \leqslant \boldsymbol{b},\ \boldsymbol{x} \geqslant \boldsymbol{0}\}$ and $\mathcal{Y} := \{\boldsymbol{y} \in \mathbb{R}^p : \exists\ \boldsymbol{x} \in \mathcal{X},\ \boldsymbol{y} = D\boldsymbol{x} + \boldsymbol{d},\ \boldsymbol{y} \geqslant \boldsymbol{0}\}$ represent the *feasible set in the decision space* and the *feasible set in the criterion space*, respectively. 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).

Assumption 4.1. We assume that $\pi_i > 0$ and it is not necessarily integer for all $i \in \{1, \dots, p\}$.

Assumption 4.2. 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 $\boldsymbol{y} > \boldsymbol{0}$.

As an aside, we note that the assumption that there exists a $\boldsymbol{y} \in \mathcal{Y}$ such that $\boldsymbol{y} > \boldsymbol{0}$ is not restrictive as it can be checked in advance due to the fact that y_i will naturally take integer values for any $i \in \{1, \dots, p\}$ in Problem (4.1) when D and $\boldsymbol{d}$ contain only integer entries/components. So, one can solve a feasibility problem in advance to identify whether there exists $\boldsymbol{y} \in \mathcal{Y}$ such that $y_i \geqslant 1$ for all $i \in \{1, \dots, p\}$.

As mentioned earlier, an IL-GMMP can be reformulated as a mixed integer SOCP if π_i is a positive integer for all $i = 1, \ldots, p$ [51]. 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 (4.1) can be reformulated as follows,

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

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 (4.1). Let k be the smallest integer value such that $2^k \geqslant \sum_{i=1}^{p}\pi_i$. By introducing a set of non-negative variables and constraints, the geometric mean constraint can be replaced as follows:

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

The above formulation is a mixed integer SOCP since any constraint of the form $\{u, v, w \geqslant 0 : u \leqslant \sqrt{vw}\}$ is equivalent to $\{u, v, w \geqslant 0 : \sqrt{u^2 + (\frac{v-w}{2})^2} \leqslant \frac{v+w}{2}\}$. Now, one can use a commercial mixed integer 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 4.1. A feasible solution $\boldsymbol{x} \in \mathcal{X}$ is called *efficient*, if there is no other $\boldsymbol{x}' \in \mathcal{X}$ such that

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

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 4.1. *An optimal solution of Problem* (4.1)*, 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 (4.1) 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

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

where $\boldsymbol{y} := D\boldsymbol{x}+\boldsymbol{d}$. In addition, by assumptions of Problem (4.1), we know that $\pi_i > 0$ for all $i = 1, ..., p$ 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). $\square$

Proposition 4.1 indicates that

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

and therefore, as we discussed earlier, an IL-GMMP is a special case of the problem of optimization over the efficient set [64]. Hence, instead of searching the entire feasible set, we propose algorithms which look for an optimal point of Problem (4.1) by searching over the set of nondominated points. We next present a critical proposition and corollary motivated by the study of [3] that result in generating effective cuts during the course of our proposed algorithms.

Proposition 4.2. *For all $\bar{\boldsymbol{y}} > \mathbf{0}$ and $\boldsymbol{\pi} > \mathbf{0}$, $\bar{\boldsymbol{y}}$ is the unique optimal point of the following L-GMMP,*

$$\max\Big\{\prod_{i=1}^{p} y_i^{\pi_i}:\ \boldsymbol{y} \geqslant \mathbf{0},\ \sum_{i=1}^{p}\frac{\pi_i}{\bar{y}_i}y_i \leqslant \sum_{i=1}^{p}\pi_i\Big\}.$$

Proof. It is known that every L-GMMP has a unique optimal point [6]. 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,

$$\boldsymbol{y}^* \in \arg\min\Big\{-\sum_{i=1}^{p}\pi_i\log(y_i):\ \boldsymbol{y} \geqslant \mathbf{0},\ \sum_{i=1}^{p}\frac{\pi_i}{\bar{y}_i}y_i \leqslant \sum_{i=1}^{p}\pi_i\Big\}.$$

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

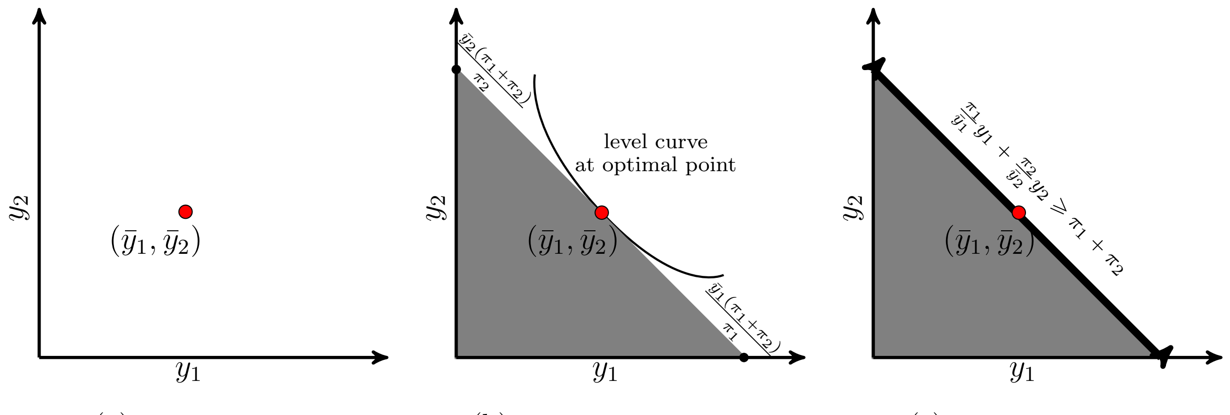
$$\boldsymbol{y}^*_R \in \arg\min\Big\{-\sum_{i=1}^{p}\pi_i\log(y_i):\ \sum_{i=1}^{p}\frac{\pi_i}{\bar{y}_i}y_i - \sum_{i=1}^{p}\pi_i \leqslant 0\Big\}.$$

The KKT conditions of the relaxed problem are as follows,

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

where η is the dual variable associated with constraint $\sum_{i=1}^{p}\frac{\pi_i}{\bar{y}_i}y_i - \sum_{i=1}^{p}\pi_i \leqslant 0$. It is 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, $\boldsymbol{y}^* = \bar{\boldsymbol{y}}$. $\square$

As an aside, in Proposition 4.2, the set $\{\boldsymbol{y} \in \mathbb{R}^p : \boldsymbol{y} \geqslant \boldsymbol{0},\ \sum_{i=1}^{p}\frac{\pi_i}{\bar{y}_i}y_i \leqslant \sum_{i=1}^{p}\pi_i\}$ can be viewed as a right-angled triangle when $p = 2$ and its generalization (for higher dimensions) when $p \geqslant 2$. 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. An illustration of Proposition 4.2 can be found in Figure 4.1 when $p = 2$. Specifically, in Figure 4.1a, a feasible point $\bar{\boldsymbol{y}} > \boldsymbol{0}$ in the criterion space is shown and in Figure 4.1b, the triangle for which $\bar{\boldsymbol{y}}$ is optimal for a given vector of geometric weights $\boldsymbol{\pi} > \boldsymbol{0}$ is shown where the level curve basically represents $\prod_{i=1}^{p} y_i^{\pi_i} = \prod_{i=1}^{p} \bar{y}_i^{\pi_i}$.



y_2
$(\bar{y}_1, \bar{y}_2)$
y_1
$\frac{\bar{y}_2(\pi_1+\pi_2)}{\pi_2}$
level curve
at optimal point
$(\bar{y}_1, \bar{y}_2)$
$\frac{\bar{y}_1(\pi_1+\pi_2)}{\pi_1}$
y_2
y_1
$\frac{\pi_1}{\bar{y}_1}y_1 + \frac{\pi_2}{\bar{y}_2}y_2 \geqslant \pi_1 + \pi_2$
$(\bar{y}_1, \bar{y}_2)$
y_2
y_1

(a) A feasible point (b) The dominated triangle (c) The hypotenuse cut

Figure 4.1: An illustration of Proposition 4.2 and its corollary when $p = 2$

Corollary 4.1. *Let $\bar{\boldsymbol{y}} > \boldsymbol{0}$ and $\boldsymbol{y}^*$ represent a feasible point and the optimal point of an IL-GMMP (with feasible set $\mathcal{Y}$), respectively. If $\mathcal{Y} \subseteq \big\{\boldsymbol{y} \in \mathbb{R}^p : \boldsymbol{y} \geqslant \boldsymbol{0},\ \sum_{i=1}^{p}\frac{\pi_i}{\bar{y}_i}y_i \leqslant \sum_{i=1}^{p}\pi_i\big\}$ then we have that $\boldsymbol{y}^* = \bar{\boldsymbol{y}}$. Otherwise, if $\mathcal{Y} \nsubseteq \big\{\boldsymbol{y} \in \mathbb{R}^p : \boldsymbol{y} \geqslant \boldsymbol{0},\ \sum_{i=1}^{p}\frac{\pi_i}{\bar{y}_i}y_i \leqslant \sum_{i=1}^{p}\pi_i\big\}$ then either $\boldsymbol{y}^* = \bar{\boldsymbol{y}}$ or $\boldsymbol{y}^* \in \big\{\boldsymbol{y} \in \mathcal{Y} : \sum_{i=1}^{p}\frac{\pi_i}{\bar{y}_i}y_i > \sum_{i=1}^{p}\pi_i\big\}$.*

Corollary 4.1 suggests that, for any strictly positive feasible point $\bar{\boldsymbol{y}} \in \mathcal{Y}$, the following cut (or half space),

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

can be added to an IL-GMMP because this inequality does not cut off its optimal point. We sometimes refer to this inequality as the *hypotenuse cut.* An illustration of the hypotenuse cut when $p = 2$ can be found in Figure 4.1c when $p = 2$. Note that one can write the proposed cut in the form of strict inequality, i.e. $>$, based on Corollary 4.1 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 4.1. *For a vector of non-negative real numbers* $(c_1, \dots, c_p)$ *and a vector of positive real numbers* $(\pi_1, \dots, \pi_p)$*, the following inequality,*

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

holds.

Proof. The proof of this theorem is based on the *Arithmetic Mean-Geometric Mean* (AM-GM) inequality [65]. For a vector of non-negative real numbers $(c_1, \dots, c_p)$ and a vector of positive real numbers $(w_1, \dots, w_p)$ with $\sum_{i=1}^{p} w_i = 1$, the following inequality,

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

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 $i \in \{1, \dots, p\}$, we have that,

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

Hence,

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

and the result follows. □

Theorem 4.1 implies the following critical remark for computing dual and primal bounds for IL-GMMPs.

Remark 4.1. If $\bar{\boldsymbol{y}}$ is an optimal point of the following integer linear program,

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

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

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

hold.

4.2 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 assumed that $\mathcal{X}$ is bounded. An illustration of the nondominated frontier when $p = 2$ can be found in Figure 4.2a.

In each iteration, the proposed algorithms attempt to compute a nondominated point (if possible) in the criterion space, denoted by $\bar{\boldsymbol{y}}$, using a specific operation. This operation is specifically designed based on Remark 4.1 and its details are given in Subsection 4.2.2.

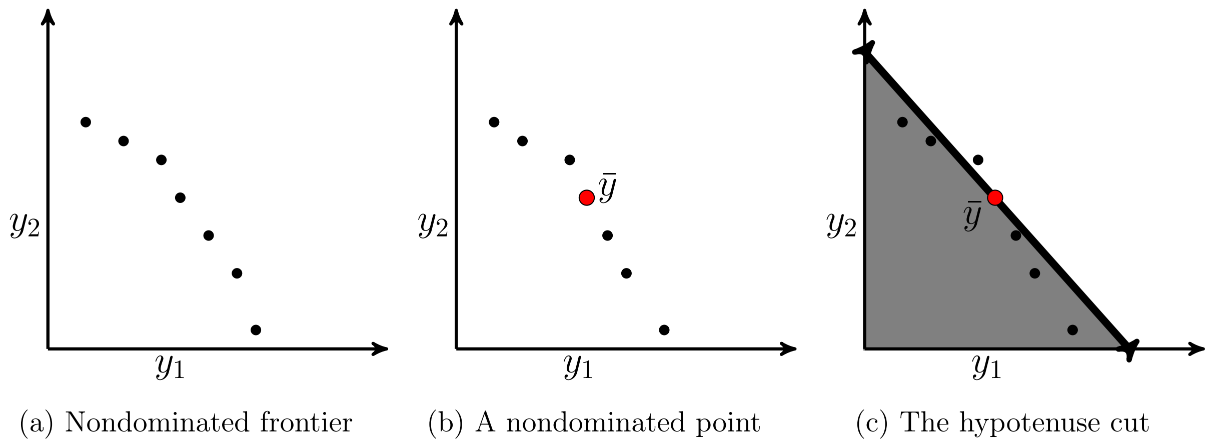


Figure 4.2: An illustration of the workings of the proposed algorithms in the first iteration when $p = 2$

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 4.2b.

Next, the proposed algorithms add the so-called *no-good constraint* to the formulation (see Section 4.2.1 for details of no-good constraint). The purpose of the no-good constraint is to ensure that (only) 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. Note that a no-good constraint may not have an impact on the feasible set in the criterion space (in a single iteration) because there may exist multiple solutions in the decision space whose image in the criterion space is exactly $\bar{\boldsymbol{y}}$ and the algorithms will only remove the one that they obtained (in a single iteration). The algorithms (if $\bar{\boldsymbol{y}} > \boldsymbol{0}$) 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 the hypotenuse cut can be found in Figure 4.2c where $\bar{\boldsymbol{y}}$ is the optimal point for the triangle and hence the triangle (except its hypotenuse) is removed from the search.

The algorithms then repeat the process explained above in the remaining feasible region and will eventually find a solution that is optimal for Problem (4.1), i.e. an IL-GMMP, with respect to a given user-specified optimality gap tolerance. 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. Note that we later, in Section 4.3, show the correctness of our algorithms through Proportions 4.3-4.5.

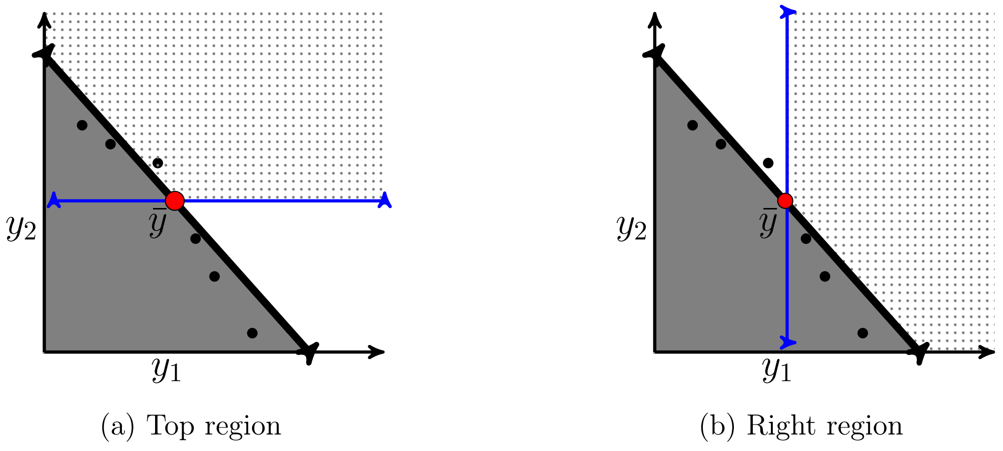


Figure 4.3: An illustration of a possible search strategy

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 4.2c 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 4.3 in which the remaining search region in Figure 4.2c is decomposed into two parts including top and right regions. In Figure 4.3 the top and right regions are created based on the observation that the region dominated by a nondominated point $\bar{\boldsymbol{y}}$, i.e. $\{\boldsymbol{y} \geqslant \boldsymbol{0} : \boldsymbol{y} \leqslant \bar{\boldsymbol{y}}\}$, can be removed from the search region. Next, the key operations used in our algorithms are explained.

4.2.1 No-good Constraint

As mentioned earlier in this section, no-good constraints, also known as *tabu* constraints [56, 59, 60], play an important role in our proposed algorithms. The underlying idea of a no-good constraint is to exclude a given feasible solution $\boldsymbol{x}' \in \mathcal{X}$ from the search (in the future iterations). With this in mind, if $\mathcal{X} \subseteq \{0, 1\}^n$, i.e., $n_i = 0$ or $I = \varnothing$, then a no-good constraint can be stated a follows,

$$\sum_{j\in\mathcal{S}^0(\boldsymbol{x}')} x_j + \sum_{j\in\mathcal{S}^1(\boldsymbol{x}')} (1-x_j) \geqslant 1, \tag{4.2}$$

where $\mathcal{S}^0(\boldsymbol{x}') := \{j \in \mathcal{B} : x'_j = 0\}$ and $\mathcal{S}^1(\boldsymbol{x}') := \{j \in \mathcal{B} : x'_j = 1\}$. This implies that no-good constraints for problems with only binary decision variables are naturally linear. Unfortunately, for problems with general integer decision variables, no-good constraints are naturally non-linear [57]. 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. $\mathcal{I} = \varnothing$ 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. $x \leqslant u$. For example, by maximizing the value of x over the linear programming relaxation of the feasible set in the decision space, i.e., $\mathcal{X}$ without the integrality condition, a global bound can be computed. 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,

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

In light of the above, 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 a no-good constraint of the form (4.2), to ensure that $\boldsymbol{x}'$ will be excluded from their search in the future iterations.

4.2.2 Weighted Sum Operation

Following Proposition 4.1, any efficient solution whose image in the criterion space lies in middle section of the nondominated frontier (and not its endpoints) is intuitively expected to be a high-quality feasible solution for Problem (4.1). So, such a solution is expected to provide a good (global) lower/primal bound for the problem. Therefore, the proposed algorithms attempt to compute such 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 result in multi-objective optimization (see for instance [49]): optimizing any positive weighted summation of the objective functions over a non-empty feasible set must return an efficient solution of that particular feasible set. In light of this observation, in this research, we simply assume that the weights are given 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 there exists any) must be 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 WSO($\boldsymbol{w}$, Pool, $\boldsymbol{l}$), can be stated as follows,

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

where $\boldsymbol{l} \geqslant \boldsymbol{0}$ 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 4.2. Due to Remark 4.1, after solving WSO($\boldsymbol{\pi}, \varnothing, \boldsymbol{0}$), global primal and dual bounds for Problem (4.1), i.e. $\max_{\boldsymbol{y} \in \mathcal{Y}} \prod_{i=1}^{p} y_i^{\pi_i}$, can be computed in linear time,

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

Similarly, after solving WSO($\boldsymbol{\pi}$, 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 WSO($\boldsymbol{\pi}$, Pool, $\boldsymbol{l}$) in the criterion space, can be computed in linear time,

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

4.3 Proposed Algorithms

In this section, we provide a detailed description of our proposed algorithms and the main reasons for developing them.

4.3.1 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 $(\boldsymbol{x}^*, \boldsymbol{y}^*)$.

In each iteration, CFSSA-I first checks whether $\mathrm{G_{UB}} - \mathrm{G_{LB}} \geqslant \varepsilon_1$ and $\frac{\mathrm{G_{UB}} - \mathrm{G_{LB}}}{\mathrm{G_{UB}}} \geqslant \varepsilon_2$ 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 $(\boldsymbol{x}^*, \boldsymbol{y}^*)$. In other words, if the WSO operation returns *null* value, the restricted region is empty of feasible solutions meaning that the current best solution is optimal. 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 4.2. 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 4 and its correctness is shown in Proposition 4.3.

```
Input: A feasible instance of Problem (4.1)
$List.create$(Pool)
$\mathrm{G_{LB}} \leftarrow -\infty$; $\mathrm{G_{UB}} \leftarrow +\infty$
$SearchDone \leftarrow$ False
while $SearchDone =$ False do
    if $\mathrm{G_{UB}} - \mathrm{G_{LB}} \geqslant \varepsilon_1$ & $\frac{\mathrm{G_{UB}}-\mathrm{G_{LB}}}{\mathrm{G_{UB}}} \geqslant \varepsilon_2$ then
        $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \leftarrow \mathrm{WSO}(\boldsymbol{\pi}, \text{Pool}, \boldsymbol{0})$
        if $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \neq$ (null,null) then
            Pool.$add((\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}))$
            if $\prod_{i=1}^{p} \bar{y}_i^{\pi_i} > \mathrm{G_{LB}}$ then
                $(\boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow (\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$
                $\mathrm{G_{LB}} \leftarrow \prod_{i=1}^{p} \bar{y}_i^{\pi_i}$
            end
            $\mathrm{G_{UB}} \leftarrow \left(\frac{\sum_{i=1}^{p}\pi_i \bar{y}_i}{\sum_{i=1}^{p}\pi_i}\right)^{\sum_{i=1}^{p}\pi_i}$
        end
        else
            $SearchDone \leftarrow$ True
        end
    end
    else
        $SearchDone \leftarrow$ True
    end
end
return $(\boldsymbol{x}^*, \boldsymbol{y}^*)$
```

Algorithm 4: CFSSA-I

Proposition 4.3. *CFSSA-I solves a finite number of single-objective integer linear programs to compute a solution that is optimal for an IL-GMMP with respect to a given user-specified optimality gap tolerance.*

Proof. First note that hypotenuse cuts are valid by Corollary 4.1 and hence they do not (negatively) impact the correctness of CFSSA-I. So, we do not consider them in this proof. With this in mind, observe that the number of feasible solutions of an IL-GMMP is finite because $\mathcal{X}$ is bounded by Assumption 4.2. In each iteration, CFSSA-I finds a feasible point/solution by solving an integer linear program through calling the weighted sum operation. Also, in each iteration the obtained solution will be excluded from the search because the algorithm adds it to the tabu list, i.e. Pool. Note that for each solution in Pool, its corresponding no-good constraint will be added to the formulation of the weighted sum operation. So, in the worst case scenario, CFSSA-I will enumerate all feasible solutions before finding an optimal solution. So, the result follows. $\square$

4.3.2 The Criterion Feasible Set Shrinking Algorithm II

We start this subsection by providing an observation about CFSSA-I, i.e. Algorithm 4. 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 $\text{WSO}(\boldsymbol{\pi}, \text{Pool}, \mathbf{0})$ 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 $\bar{\boldsymbol{y}} \not> \mathbf{0}$ does not occur, i.e. none of the components of the vector $\bar{\boldsymbol{y}}$ can be zero. This is because based on Assumption 4.2 the optimal objective value of an IL-GMMP is strictly positive. Hence, one can simply add the condition $y_i \geqslant \epsilon$ for all $i = 1, \ldots, p$ to the formulation of an IL-GMMP where ϵ is a sufficiently small positive value. Adding these constraints guarantees that only feasible points with strictly positive objective values will be produced during the course of our algorithms. Note that one way to compute ϵ is to solve a max-min optimization problem, i.e. $\epsilon = \max_{\boldsymbol{y}\in\mathcal{Y}} \min(y_1, \ldots, y_p)$.

We first note that at any iteration of Algorithm 4, after calling WSO($\boldsymbol{\pi}$, POOL, $\mathbf{0}$), $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ will be returned and will be added to POOL if $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \neq (null, null)$. So, we know that at the next iteration of Algorithm 4, the following hypotenuse cut,

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

exists in the model corresponding to WSO($\boldsymbol{\pi}$, POOL, $\mathbf{0}$). Observe that, as mentioned in Section 4.1, we could write this cut in the form of strict inequality based on Corollary 4.1. Hence, if we use the strict inequality, the feasible point $\bar{\boldsymbol{y}}$ can never be computed again in any iteration of Algorithm 4 and this resolves the issue that we mentioned above.

```
Input: A feasible instance of Problem (4.1)
List.create(POOL)
$G_{LB} \leftarrow -\infty$; $G_{UB} \leftarrow +\infty$
SearchDone ← FALSE
while SearchDone = FALSE do
    if $G_{UB} - G_{LB} \geqslant \varepsilon_1$ & $\frac{G_{UB}-G_{LB}}{G_{UB}} \geqslant \varepsilon_2$ then
        $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \leftarrow$ WSO($\boldsymbol{\pi}$, POOL, $\mathbf{0}$)
        if $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \neq$ (null,null) then
            POOL.add($(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$)
            if $\prod_{i=1}^{p} \bar{y}_i^{\pi_i} > G_{LB}$ then
                $(\boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow (\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$
                $G_{LB} \leftarrow \prod_{i=1}^{p} \bar{y}_i^{\pi_i}$
            end
            $G_{UB} \leftarrow \left(\frac{\sum_{i=1}^{p} \pi_i \bar{y}_i}{\sum_{i=1}^{p} \pi_i}\right)^{\sum_{i=1}^{p} \pi_i}$
            if $\bar{\boldsymbol{y}} > \mathbf{0}$ then
                $\hat{\boldsymbol{\pi}} \leftarrow (\frac{\pi_1}{\bar{y}_1}, \ldots, \frac{\pi_p}{\bar{y}_p})$
                $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \leftarrow$ WSO($\hat{\boldsymbol{\pi}}$, POOL, $\mathbf{0}$)
                if $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$ =(null, null) or $\sum_{i=1}^{p} \hat{\pi}_i \hat{y}_i \leqslant \sum_{i=1}^{p} \pi_i$ then
                    SearchDone ← TRUE
                end
                else
                    if $\prod_{i=1}^{p} \hat{y}_i^{\pi_i} > G_{LB}$ then
                        $(\boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow (\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$
                        $G_{LB} \leftarrow \prod_{i=1}^{p} \hat{y}_i^{\pi_i}$
                    end
                    POOL.add($(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$)
                end
            end
        end
        else
            SearchDone ← TRUE
        end
    end
    else
        SearchDone ← TRUE
    end
end
return $(\boldsymbol{x}^*, \boldsymbol{y}^*)$
```

Algorithm 5: CFSSA-II

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 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 5 which is similar to Algorithm 4 but has some additional lines, i.e. see Lines 14-23. Also, the correctness of CFSSA-II is shown in Proposition 4.4.

Proposition 4.4. *CFSSA-II solves a finite number of single-objective integer linear programs to compute a solution that is optimal for an IL-GMMP with respect to a given user-specified optimality gap tolerance.*

Proof. Similar to the proof of Proposition 4.3. The only difference is that in each iteration one additional weighted sum operation may be called. □

4.3.3 The Criterion Feasible Set Shrinking Algorithm III

The third algorithm, the so-called CFSSA-III, is a significantly modified/enhanced version of the algorithm proposed by [1]. The two main differences are as follows. First, the hypotenuse cut does not exist in the algorithm proposed by [1]. Second, to compute a dual bound at each iteration, the algorithm proposed by [1] 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 [1] 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 [1] cannot directly handle non-unit geometric weights as the size of the SOCP reformulation highly depends on the geometric weights.

CFSSA-III maintains a queue of nodes, denoted by $\textsc{Tree}$. 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 $\boldsymbol{l} = \boldsymbol{0}$ and $\text{UB} = +\infty$. Similar to CFSSA-I and CFSSA-II, CFSSA-III maintains three other pieces of information including $\textsc{Pool}$, G_{UB}, and G_{LB}. At the beginning, $\textsc{Pool}$ 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 $\text{G}_{\text{UB}} - \text{G}_{\text{LB}} \geqslant \varepsilon_1$ and $\frac{\text{G}_{\text{UB}} - \text{G}_{\text{LB}}}{\text{G}_{\text{UB}}} \geqslant \varepsilon_2$. 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 $\text{WSO}(\boldsymbol{\pi}, \textsc{Pool}, \boldsymbol{l})$ 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 $\textsc{Pool}$.
- 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 a chance of finding a better solution in this node, i.e. $\text{UB} - \text{G}_{\text{LB}} \geqslant \varepsilon_1$ and $\frac{\text{UB}-\text{G}_{\text{LB}}}{\text{UB}} \geqslant \varepsilon_2$, 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 $l^i_i = \bar{y}_i$.

```
Input: A feasible instance of Problem (4.1)
$Queue.create$(TREE)
$List.create$(POOL)
$\boldsymbol{l} \leftarrow \boldsymbol{0}$; 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) & $SearchDone$ = FALSE do
    TREE.$PopOut(\boldsymbol{l}, \text{UB})$
    $\text{G}_{\text{UB}} \leftarrow \text{UB}$
    if $\text{G}_{\text{UB}} - \text{G}_{\text{LB}} \geqslant \varepsilon_1$ & $\frac{\text{G}_{\text{UB}}-\text{G}_{\text{LB}}}{\text{G}_{\text{UB}}} \geqslant \varepsilon_2$ 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
            $\text{UB} \leftarrow \left(\frac{\sum_{i=1}^{p} \pi_i \bar{y}_i}{\sum_{i=1}^{p} \pi_i}\right)^{\sum_{i=1}^{p} \pi_i}$
            POOL.$add((\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}))$
            if $\prod_{i=1}^{p} \bar{y}_i^{\pi_i} > \text{G}_{\text{LB}}$ then
                $(\boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow (\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$
                $\text{G}_{\text{LB}} \leftarrow \prod_{i=1}^{p} \bar{y}_i^{\pi_i}$
            end
            if $\text{UB} - \text{G}_{\text{LB}} \geqslant \varepsilon_1$ & $\frac{\text{UB}-\text{G}_{\text{LB}}}{\text{UB}} \geqslant \varepsilon_2$ then
                foreach $i \in \{1, 2, .., p\}$ do
                    $\boldsymbol{l}^i \leftarrow \boldsymbol{l}$
                    $l^i_i \leftarrow \bar{y}_i$
                    TREE.$add(\boldsymbol{l}^i, \text{UB})$
                end
            end
        end
    end
    else
        $SearchDone \leftarrow$ TRUE
    end
end
return $(\boldsymbol{x}^*, \boldsymbol{y}^*)$
```

Algorithm 6: CFSSA-III

In light of the above, a precise description of CFSSA-III can be found in Algorithm 6 and its correctness is shown in Proposition 4.5. 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.

Proposition 4.5. *CFSSA-III solves a finite number of single-objective integer linear programs to compute a solution that is optimal for an IL-GMMP with respect to a given user-specified optimality gap tolerance.*

Proof. Similar to the proof of Proposition 4.3. The only difference is that CFSAA-III decomposes the search region of each iteration into at most p (smaller) search regions (rather than 1) at the end of each iteration. By construction, the union of the search regions is exactly the original feasible set. Also, because the tabu list, i.e. Pool, is defined globally in CFSSA-III, any feasible solution found in each iteration cannot be found in any subsequent search regions. So, in the worst case scenario, CFSSA-III will enumerate all feasible solutions before finding an optimal solution. □

4.4 Computational Study

In this section, we conduct an extensive computational study using three commercial solvers including CPLEX 12.7, Gurobi 8.1.1, and Xpress 8.5.6. 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 4.1, 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://github.com/paymanghasemi. 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 7.0 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 instances and solve each one using different geometric weights shown in Table 4.1. 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. Note that in our computational study, we do not generate instances with $p > 4$ to avoid numerical instability. Theoretically, all solution methods discussed in the paper should handle any IL-GMMPs with any arbitrary value of p. However, in practice, as p increases, the optimal objective value of Problem (4.1) tends to grow dramatically which directly or indirectly result in numerical instabilities in all existing solution methods/solvers. Developing custom-built exact algorithms for solving IL-GMMPS that are guaranteed to not run into objective-value-caused numerical issues for large values of p is currently an active research of the authors.

Table 4.1: Different geometric weights for each value of p

	p = 4	**p = 3**	**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 $\leqslant$ inequality and all coefficients of matrix A are nonnegative, the set $\mathcal{X}$ is bounded.

As mentioned in Section 4.2, 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) the 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 take a maximum value of 15. Therefore, the integer variable x_i is replaced by four binary decision variables and one additional constraint as follows,

$$x_i := z_{i_1} + 2z_{i_2} + 4z_{i_3} + 8z_{i_4},$$
$$z_{i_1} + 2z_{i_2} + 4z_{i_3} + 8z_{i_4} \leqslant 15.$$

Note that the additional constraint is redundant and there is no need to add it to the problem. 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 same instance. Hence, if the ratio is closer to one, then it is better. Another tool that we use is *performance profile* [52] 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.

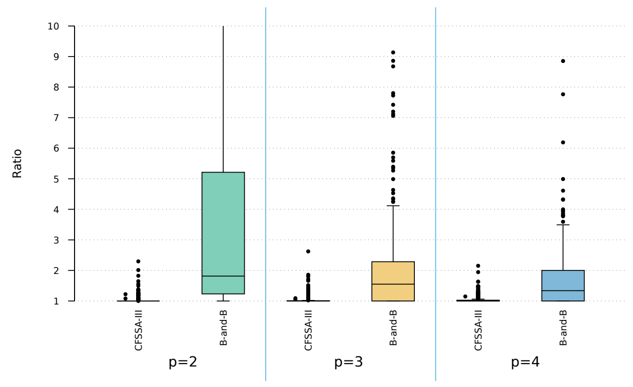


Figure 4.4: Comparison between the algorithm proposed by [1], the so-called B-and-B, and CFSSA-III on pure binary instances for unit geometric weights

As mentioned in the Introduction and Section 4.3.3, CFSSA-III is a modified/enhanced version of the algorithm proposed by [1], the so-called *B-and-B*. Figure 4.4 is a boxplot for comparing the performance of these two algorithms on pure binary instances. Note that each segment in the figure is independent of others as they represent different classes of instances (due to the difference in their values of p). In other words, in each segment, the ratios are calculated based on the minimum of the solution times reported by only the two algorithms listed in the segment. We also note that the C++ implementation of B-and-B is publicly available and uses CPLEX. Moreover, B-and-B cannot deal with the non-unit geometric weights directly and similar to CFSSA-III 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 4.4 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 the randomly generated instances used in our computational study.

4.4.1 Two Objectives ($p = 2$)

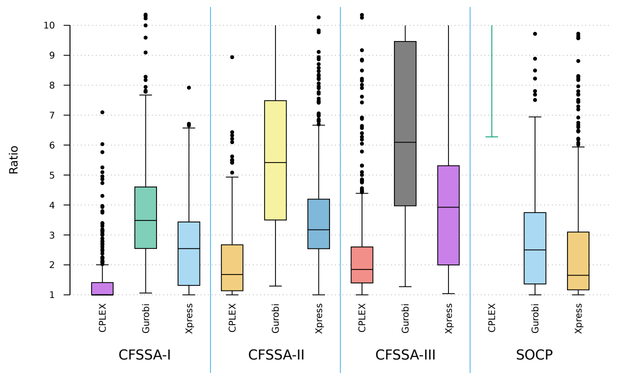
In this section, we compare the performance of the algorithms on instances with $p = 2$. CFSSA-I and CFSSA-II were able to solve all instances to optimality. However, CFSSA-III and the (mixed integer) SOCP solver of CPLEX were not able to do so even for unit geometric weights. Table 4.2 shows a comparison between the 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, the SOCP solver of CPLEX is performing the worst by reporting an average optimality gap of around 60% for more than one third of the instances.

Table 4.2: Performance comparison for instances with $p = 2$ and $\boldsymbol{\pi} = (1, 1)$

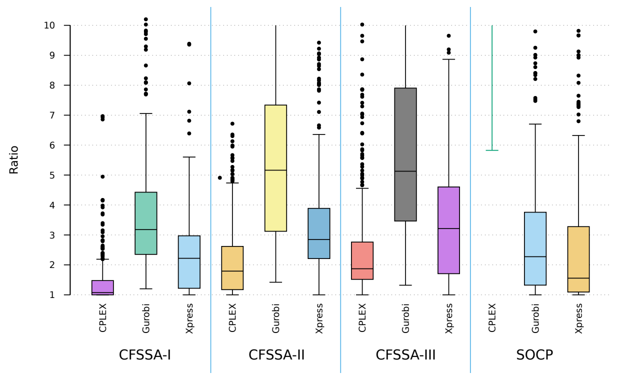
		CFSSA-III			**SOCP**		
		CPLEX	Gurobi	Xpress	CPLEX	Gurobi	Xpress
Binary	#S	400	400	400	400	400	400
	#Opt	395	392	394	249	400	400
	%Gap	0.06	0.09	0.06	58.92	0	0
Integer	#S	400	400	400	400	400	400
	#Opt	391	391	392	252	400	400
	%Gap	0.06	0.06	0.06	60.66	0	0

Figure 4.5 shows the run time boxplots of the algorithms on instances with $p = 2$ for unit geometric weights. Note that, unlike Figure 4.4, each segment in Figure 4.5 is not independent of others as the same instances are used in all segments. In other words, the ratios are calculated based on the minimum of the solution times reported by all 12 algorithms listed in the figure. From this figure, it is clear that all our proposed algorithms perform significantly better when using the mixed integer linear programming solver of CPLEX compared to Gurobi and Xpress. However, the SOCP solver of CPLEX is the worst choice for solving a mixed integer SOCP transformation of an IL-GMMP. The 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 the 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 the SOCP solver of Xpress.

Figures 4.6 and 4.7 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

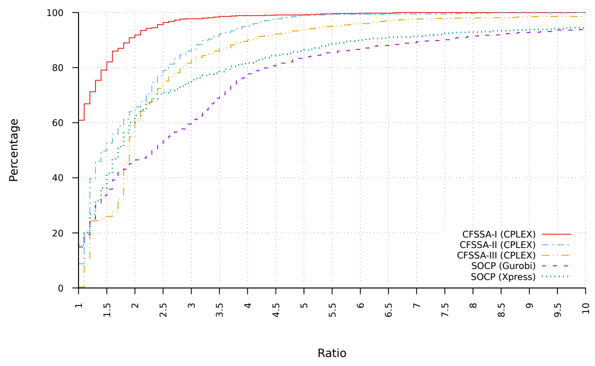


(a) Pure Binary

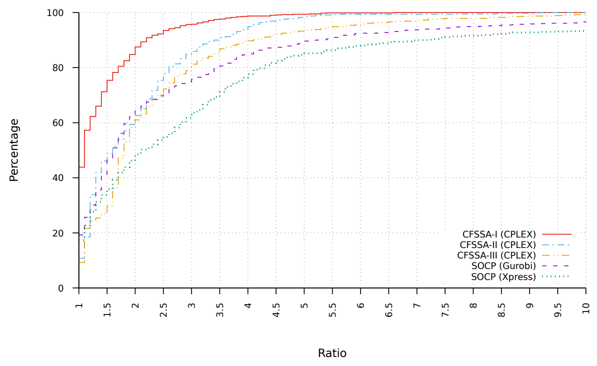


(b) Pure Integer

Figure 4.5: Performance comparison for instances with $p = 2$ and $\boldsymbol{\pi} = (1, 1)$



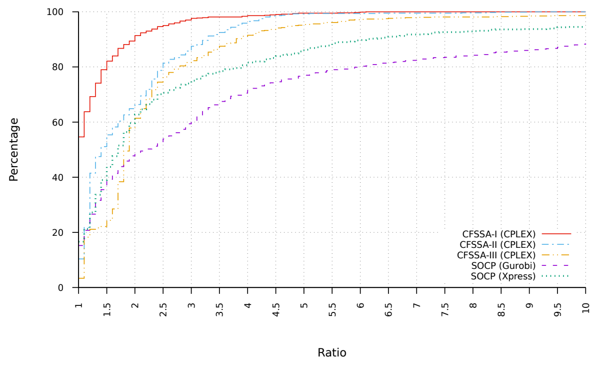
(a) Pure Binary



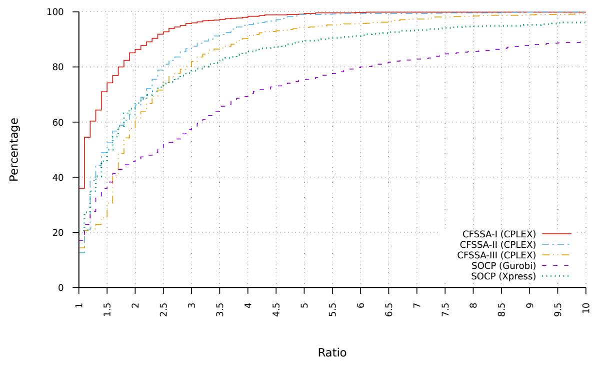
(b) Pure Integer

Figure 4.6: Performance profile for instances with $p = 2$ and $\boldsymbol{\pi} \in \{(2, 1), (1, 2)\}$

these figures. Overall, CFSSA-I (when using CPLEX) performs significantly better than the 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, it is evident that CSFSA-I solved around 20% of the instances at least 2.5 times faster than the others 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 4.6b. Overall, they are both either competitive with CFSSA-III or slightly worse than it.



(a) Pure Binary



(b) Pure Integer

Figure 4.7: Performance profile for instances with $p = 2$ and $\boldsymbol{\pi} \in \{(3,1),(1,3)\}$

4.4.2 Three Objectives ($p = 3$)

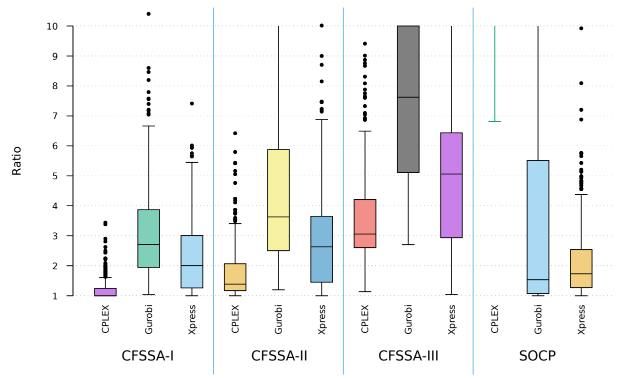
In this section, we compare the performance of the algorithms on instances with $p = 3$. Again, CFSSA-I and CFSSA-II were able to solve all instances to optimality within the imposed time limit. However, CFSSA-III and the (mixed integer) SOCP solvers of CPLEX and Xpress were not able to do so for all the instances with unit geometric weights. Table 4.3 shows a comparison between the SOCP solvers and CFSSA-III for unit geometric weights. We observe from the table that the SOCP solver of Xpress was not able to solve only one of the instances to optimality for the class of pure integer instances. The 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 an average optimality gap of 0.34%.

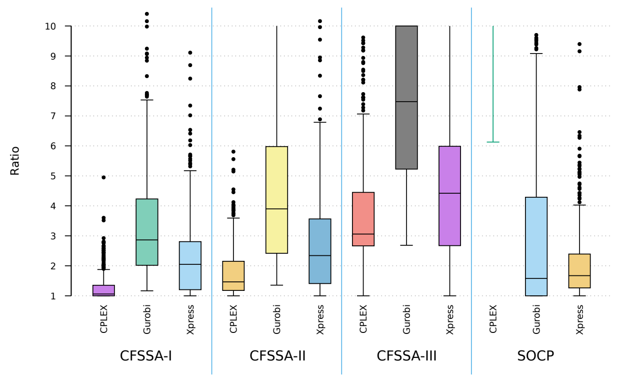
Another interesting observation is that the SOCP solver of CPLEX was not able to solve, i.e. find a feasible solution (other than zero), for 208 and 219 instances of the classes pure binary and pure integer, respectively. According to [61], the reason can be the convergence tolerance. Changing this tolerance 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 4.3: 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

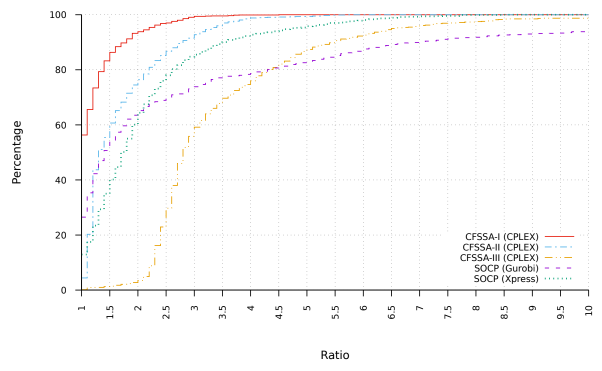


(a) Pure Binary ($n = n_b$)

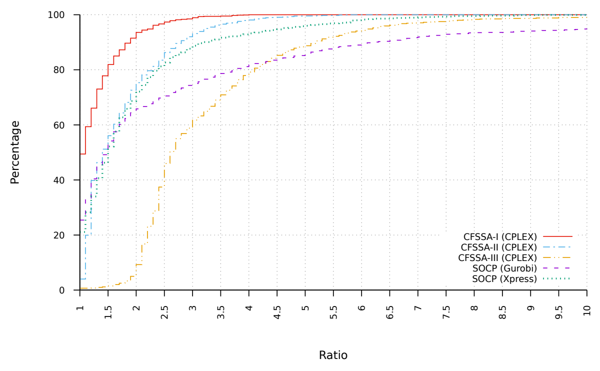


(b) Pure Integer ($n = n_i$)

Figure 4.8: 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 4.9: Performance profiles for instances with $p = 3$ and $\boldsymbol{\pi} \in \{(2,1,1),(1,2,1),(1,1,2)\}$

Figure 4.8 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, the SOCP solver of CPLEX is the worst choice for solving a mixed integer SOCP transformation of an IL-GMMP. The 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 the SOCP solver of Xpress by a factor of around 1.5. Finally, CFSSA-II (when using CPLEX) is competitive with the SOCP solver of Xpress but CFSSA-III is not.

Figure 4.9 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, the 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 4.7), 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.

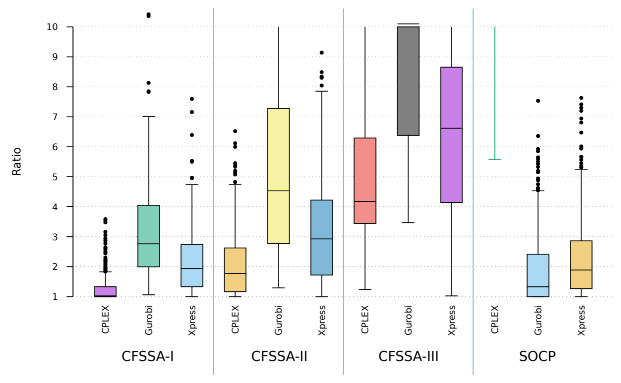
4.4.3 Four Objectives ($p = 4$)

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 of CPLEX and Xpress were not able to do so for all the instances with unit geometric weights. Table 4.4 shows a comparison between the SOCP solvers and CFSSA-III for unit geometric weights. We observe from the table that the SOCP solver of Gurobi was able to solve all instances to optimality. However, the 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, the 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.

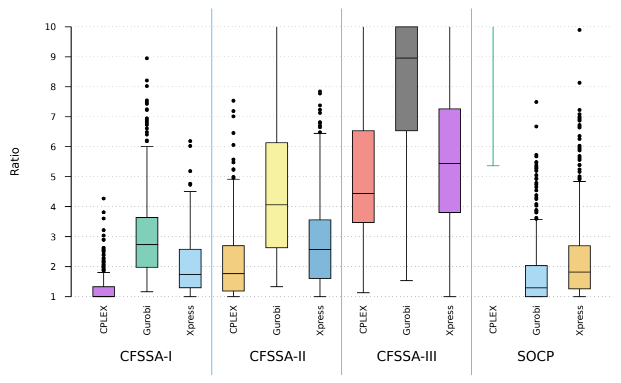
Table 4.4: Performance comparison for instances with $p = 4$ and $\boldsymbol{\pi} = (1, 1, 1, 1)$

		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

Figure 4.10 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 the mixed integer linear programming solver of CPLEX.



(a) Pure Binary ($n = n_b$)



(b) Pure Integer ($n = n_i$)

Figure 4.10: Performance comparison for instances with $p = 4$ and $\boldsymbol{\pi} = (1, 1, 1, 1)$

However, this time, although the SOCP solver of Xpress performs well, it is not as good as the SOCP solver of 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 the 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 the SOCP solver of Xpress.

4.4.4 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 the easiest case is linearizing pure binary instances with $p = 2$ and $\boldsymbol{\pi} = (1, 1)$. 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,

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

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

$$\begin{aligned}
\max\ & 3x_2 + q_1 + 2q_2 + 6q_3 & \\
\text{s.t. } & q_1 \leqslant x_1, \ q_1 \leqslant x_2, & (x_1 x_2) \\
& q_2 \leqslant x_1, \ q_2 \leqslant x_3, & (x_1 x_3) \\
& q_3 \leqslant x_2, \ q_3 \leqslant x_3, & (x_2 x_3) \\
& q_i \in \{1, 0\} & \forall i \in \{1, 2, 3\}
\end{aligned}$$

We linearized all pure binary instances of Section 4.4.1 in the way discussed above and solved them by CPLEX, Gurobi, and Xpress. Table 4.5 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 the largest classes of instances to optimality in around 200 seconds on average. Interested readers may refer to Appendix C1 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 will be even worse for other classes or larger values of p and $\boldsymbol{\pi}$.

Table 4.5: Detailed performance of single-objective integer linear programming solvers on linearized binary instances with $p = 2$ 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

Chapter 5: A Cooperative Game Solution Approach for Intensity Modulated Radiation Therapy Design: Nash Social Welfare Optimization

The copyright permissions for reuse previously published material in this chapter can be found in Appendix A3.

5.1 Introduction

Cancerous tissues are fast proliferating cells that are more sensitive to radiation compared to healthy cells. This fact provides the basis to fight against cancers using radiotherapy. In radiotherapy, radiation doses are delivered to cancerous cells which helps shrink or eliminate the tumors. One of the radiotherapy methods is Intensity Modulated Radiation Therapy (IMRT) that uses computer-controlled accelerators to deliver radiation doses to a tumor or specific areas within the tumor. IMRT controls the intensity of the radiation beam in multiple small volumes and helps the radiation dose to conform to the 3D shape of the target area. In IMRT, planning is a critical problem which concerns with the choice of the best setting of radiation. In IMRT planning, the goal is to select the emission plan that assures the deliverance of tumoricidal radiation doses to Planning Target Volume (PTV) with the minimal impact on healthy organs, referred to as Organs At Risk (OAR). The planning is usually divided into three phases [66], namely

1. the selection of the radiation angles and beams (*beam angle optimization*),
2. the design of fluency map or intensity pattern (*fluency map optimization*),
3. the design of a delivery sequence (*segmentation problem*).

In this research, our focus is on the second step, i.e., fluency map optimization. Our motivation is that, although the entire process of fluency map optimization is based on the trade-offs between killing cancerous cells and not harming healthy cells, there are no research in the literature focusing on modeling these trade-offs from the angle of cooperative game theory. Therefore, we use a game theoretical approach to create a cooperative game by solely focusing on modeling the trade-offs occurring in the fluency map optimization problem. To do so, in what follows in this section, we first provide a brief literature review of how a fluency map optimization is mathematically modeled. Then, we discuss more about our motivation and provide a brief explanation of our contributions. Finally, we provide the main structure of the remainder of our paper.

5.1.1 Literature Review

In fluency map optimization, the assumption is that a number of beam angles are provided, and the problem is to design a fluency map which maximizes the dose delivered to PTV while minimizing the doses deposited in OAR. To calculate the depositions made by the radiations of the selected beam angles, the 3D computed tomography (CT) of the patient is required. The patient's body will be considered as a net of small volume elements, referred to as Voxels, and the dose deposition will be modeled to show how much the amount of depositions in different body voxels will be if a specific beam or a group of beams radiate.

The dose deposition model is non-linear in nature [67]; however, in the literature of radiotherapy design, it is well studied that the linear dose deposition matrices can provide adequately precise approximations of the depositions [68, 69, 70, 71]. This, in specific, is of benefit as it enables modeling the constraints of the fluency map optimization using linear functions rather than non-linear relations. To present the linear deposition relations, we denote the set of all beamlets by N and the set of all body voxels by V. Further, we define $\boldsymbol{x} := (x_1, \cdots, x_{|N|})$ as the vector of variables where x_n represents the amount of radiation of beamlet $n \in N$, and we define D_{vn} as the amount of deposition in voxel $v \in V$ if beamlet

n emits one unit radiation, i.e., $x_n = 1$. We also define $\boldsymbol{d} := (d_1, \cdots, d_{|V|})$ as the vector of doses where d_v is the amount of the total doses deposited in voxel $v \in V$. With these definitions, the set of all possible fluency maps can be written as

$$\mathcal{D} := \{\boldsymbol{d} \in \mathbb{R}_+^{|V|} : d_v = \sum_{n \in N} D_{vn} x_n \quad \forall v \in V, \qquad \boldsymbol{x} \geqslant 0\}. \tag{5.1}$$

Having defined the deposition relations, a dose prescription is required to design the fluency map based on it. The dose prescription typically includes a lower and upper dose level for each organ or voxel in the body. A common problem with the clinical dose prescriptions is that designing the ideal fluency map delivering such prescriptions is almost impossible [72]. This is due to the fact that delivering the tumoricidal radiation doses to PTV often requires the violation of some of the prescribed bounds for healthy voxels. Therefore, instead of finding a plan satisfying the prescription, the fluency map optimization problem can be stated as finding a plan that minimizes the deviations from the prescribed levels. That said, the fluency map optimization problem can be stated as

$$\min_{\boldsymbol{d} \in \mathcal{D}} f(\boldsymbol{d}), \tag{5.2}$$

where $f(\boldsymbol{d})$ represents the non-negative function that measures the deviations, which can be linear, quadratic, or (convex) non-linear in general [73, 74, 71, 75]. Since the quadratic penalty functions have been considered as an accepted standard in the literature of radiotherapy planning [76, 77, 78, 66, 79, 80, 81], we focus on quadratic penalty functions but our proposed approach is generic and can be customized for any other forms.

To define the quadratic penalty function, we show the dose prescription by $(\boldsymbol{l}, \boldsymbol{u})$ and define $\boldsymbol{l} := (l_1, \cdots, l_{|V|})$ and $\boldsymbol{u} := (u_1, \cdots, u_{|V|})$ as the respective vector of lower and upper dose levels such that l_v and u_v show the lower and upper dose levels prescribed for voxel $v \in V$. In addition, we define $\boldsymbol{\alpha} := (\alpha_1, \cdots, \alpha_{|V|})$ as the vector of non-negative real numbers where α_v represents the importance of voxel $v \in V$. Using these notations/definitions, the quadratic fluency map optimization problem can be defined as

$$\min_{\boldsymbol{d}\in\mathcal{D}} f(\boldsymbol{d}) = \sum_{v\in V} \alpha_v[(d_v - u_v)_+^2 + (l_v - d_v)_+^2], \tag{5.3}$$

where $(Y)_+ := \max(0, Y)$. Observe that Problem (5.3) penalizes the doses that surpass the upper bound or fall below the lower bound by emphasizing the higher deviations.

5.1.2 Motivation

Ideally, the $(\boldsymbol{l}, \boldsymbol{u})$ is defined as

$$l_v = u_v = \begin{cases} 0 & \text{when } v \text{ is OAR} \\ T_v & \text{when } v \text{ is PTV} \end{cases}, \tag{5.4}$$

where T_v is the dose level required to eliminate the PTV. Such prescription often makes the optimal objective value of Problem (5.3) non-zero. Such a non-zero value does not provide any information other than implying that it is impossible to not violate the prescribed dose levels. This lack of interpretability in the objective values constructs the main weakness of quadratic penalty functions. Specifically, due to this weakness, the plans cannot be simply evaluated using their penalty functions' values. As a result, the fluency map plans are usually evaluated and compared using their Dose Volume Histogram (DVH), a histogram relating radiation dose to tissue volume [82]. By means of clarity, a DVH is a 2D plot where the vertical axis represents the percentage volume and the horizontal axis shows the dose amount. Then, the height of a point on the plot provides the percentage volume of the structure that receives a dose greater than or equal to the length of that point.

The second weakness of quadratic penalty functions is that they do not suffice to create a plan with a clinically acceptable DVH, and the burden is on the manipulation of the importance weights, i.e., $\boldsymbol{\alpha}$, to adjust the DVH of the final plan [78]. Their third weakness is that the importance weights have no clinical meaning and are priorly unknown, their choice is quite arbitrary, and they are patient-specific [66]. Therefore, several plans with different choice of weights have to be tried before selecting a final plan. All of these add to the complexity of the problem in terms of both time and computation.

To handle the problem of importance weights, [78] proposed an automated framework that iteratively updates the weights in voxel level, but they found some issues regarding the consistency in quality and control on the trade-offs. Moreover, if the initial dose prescription is feasible, then the objective value of Problem (5.3) will be zero making the importance weights completely useless in improving the DVH of the final plan. Therefore, instead of weights, [76] used the prescribed dose levels as the driving force. They proposed a threshold-driven penalty function, where they updated the thresholds iteratively to attain a desired plan. However, their approach is more of a re-planning or adaptive planning rather than a blank-start optimizer for IMRT. In addition, in their proposed approach, they add some quadratic terms to the original quadratic penalty function which escalates the problem of meaningless penalty values and importance weights.

In this research, we want to address these issues by employing the concept of bargaining from the field of cooperative game theory. Our main motivation comes from our observation that the entire process of IMRT planning seeks to find a solution that can desirably balance the trade-offs between sacrificing some OAR's or sparing some PTV's. However, to the best of our knowledge, there are no research in the literature focusing on finding a desirable balance from a bargaining point of view. That said, we now provide the contributions of our proposed methodology.

5.1.3 Contributions

In order to discuss the contributions of our research, we first provide a brief definition of the bargaining game in the field of game theory. The bargaining problem is a game where all (competing) players agree to create a grand coalition, instead of competing with each other, to get higher payoffs [11, 83]. To be able to create a grand coalition, the agreement of all players is necessary. Therefore, the main concern when dealing with a bargaining problem is what the payoff of each player should be in a grand coalition (and how it should be computed). One of the well-known solution techniques for this problem was first introduced by [5, 6] and then extended further by [62] and [84]. In this solution approach, a supercriterion will be optimized over the feasible allocation of payoffs. This supercriterion is known as the Nash Social Welfare (NSW) function and it guarantees both efficiency and fairness in the solution that it obtains [12, 85]. For more details, the interested readers may refer to [86].

The main contribution of our research is to transform the fluency map optimization problem into a bargaining problem. We do this transformation in organ level where different body organs are considered as different players with different negotiation powers. As an overview, our approach takes a dose prescription and a penalty function for each organ as inputs and then constructs a bargaining game between them and finds a solution for it. To be able to construct the bargaining problem, we will solve $2|S| + 1$ optimization problems where S represents the set of body organs (under consideration). Having constructed the bargaining problem, its corresponding optimization problem will be solved to find the desirable fluency map, which will be a Nash optimal solution. Overall, our proposed methodology has several advantages and resolves the weaknesses of the state-of-the-art techniques mentioned in Section 5.1.2 as explained below.

- Our methodology provides the practitioners with the flexibility of using any form of penalty functions as their inputs. However, as mentioned earlier, we only focus on quadratic penalty functions due to their popularity.

- Given the convexity of input penalty functions, all optimization problems in our methodology are convex and can be solved to optimality in polynomial times.

- Our methodology resolves the weakness of interpretability as we transform the penalty functions to preference functions. Considering that the new objective values are now meaningful, the different plans can be easily evaluated and compared with respect to their objective values.

- We introduce a new control lever in our modeling referred to as negotiation powers. These powers provide the practitioners with the flexibility of putting more emphasis on an organ by changing its negotiation power.

- Finally, although we only focus on the trade-offs in organ level and use the negotiation powers as our main lever of controlling the trade-offs, we are including voxel weights and organ weights in our model to provide the ability to control the trade-offs in all levels.

5.1.4 Structure

The remainder of this chapter is organized as follows. In Section 5.2, we provide the preliminaries of bargaining problems and how these problems can be optimized. In Section 5.3, we provide the details of our methodology by explaining each step in the process of transforming a fluency map optimization to a bargaining problem. In Section 5.4, we provide a theoretical discussion about the proposed approach from the angle of multi-criteria optimization. In Section 5.5, we provide some numerical results by implementing our approach on some instances available in the literature and generating some different plans. Finally, in Section 7.4, we conclude the paper and provide some future research directions.

5.2 Preliminaries

In this section, we will discuss the preliminaries of bargaining games. In general, to create a bargaining game, four pieces of information are required:

- The set of feasible actions available for each player: before starting bargaining, players will assess their set of actions and will join the game when they are fully aware of the actions that are feasible for them to take.

- The utility function of each player: based on their set of feasible actions, players will define a utility function for themselves, which they will try to optimize during the bargaining process.

- The disagreement point or the status quo of the game: each bargaining game has a disagreement point that indicates the payoff of each player if the negotiations break down. No player accepts a payoff worse than the one in the disagreement point.

- The negotiation powers of players: evidently, stronger players want to receive better payoffs in the final solution, and the negotiation powers help differentiate between strong and weak players.

In the context of our research, the bargaining problem is a full-information cooperative game. The 'full-information' setting means that all players know all four pieces of information about all players. The 'cooperative setting' implies that all players are willing to form a grand coalition to obtain higher payoffs compared to the status quo of the game. That being said, in this research, we let S be the set of all players (which are body organs), $\mathcal{X}$ be the set of feasible actions of all players, and $\boldsymbol{r} := (r_1, \cdots, r_{|S|})$ be the disagreement point where r_s represents the payoff of player $s \in S$ in the disagreement point. Further, we define $\boldsymbol{u}(\boldsymbol{d}) := (u_1(\boldsymbol{d}), \cdots, u_{|S|}(\boldsymbol{d}))$ as the vector of non-negative utility functions and $\boldsymbol{p} := (p_1, \cdots, p_{|S|})$ as the vector of negotiation powers where $u_s(\boldsymbol{d})$ and p_s represent the utility function and negotiation power of player $s \in S$, respectively.

5.2.1 Nash Social Welfare Optimization

In order to find a solution to the bargaining game, we need to define a super-criterion for the problem that can measure the payoff of the grand coalition. To find the optimal grand coalition, we need to optimize the super-criterion on the set of all feasible actions. Such super-criteria are often referred to as social welfares in the literature of bargaining problems. In this study, we employ the Nash Social Welfare (NSW) as it is well-known that it addresses both efficiency and fairness at the same time when being optimized [12, 85]. Since in the context of our research, each player seeks to *minimize* its utility function, the Nash solution, denoted by $\boldsymbol{d}^*$, to the bargaining problem can be obtained by solving the following optimization problem,

$$\boldsymbol{d}^* \in \arg\max \Big\{ \prod_{s \in S} [r_s - u_s(\boldsymbol{d})]^{p_s} : \boldsymbol{d} \in \mathcal{X},\ u_s(\boldsymbol{d}) \leqslant r_s\ \forall s \in S \Big\}, \tag{5.5}$$

where $\prod\limits_{s \in S} [r_s - u_s(\boldsymbol{d})]^{p_s}$ is the NSW function, and $[r_s - u_s(\boldsymbol{d}^*)]$ is the benefit that player $s \in S$ obtains as a result of creating the grand coalition. Note that constraint $u_s(\boldsymbol{d}) \leqslant r_s$ for each $s \in S$ ensures that no player accepts a payoff worse than its guaranteed payoff in the disagreement point. In other words, it guarantees that the benefits have to be non-negative. Also, the benefits are usually from different orders of magnitude, and therefore, comparing different players' benefits can be misleading. As a result, we present the following theorem which says that, by scaling the benefit of any player, an equivalent optimization problem will be created.

Theorem 5.1. *The NSW is scale-free meaning that, by replacing the objective function of Problem* (5.5) *by the following function,*

$$\prod_{s \in S} [\beta_s r_s - \beta_s u_s(\boldsymbol{d})]^{p_s},$$

an equivalent problem will be constructed if β_s is a positive constant for all $s \in S$ [86].

Following Theorem 5.1, we normalize the benefits of players so that they take values between 0 and 1 as follows,

$$\boldsymbol{d}^* \in \arg\max \Big\{ \prod_{s \in S} [\frac{r_s - u_s(\boldsymbol{d})}{r_s - m_s}]^{p_s} : \boldsymbol{d} \in \mathcal{X},\ u_s(\boldsymbol{d}) \leqslant r_s\ \forall s \in S \Big\}, \tag{5.6}$$

where

$$m_s = \min\{u_s(\boldsymbol{d}) : \boldsymbol{d} \in \mathcal{X}\}. \tag{5.7}$$

We refer to $[\frac{r_s - u_s(\boldsymbol{d})}{r_s - m_s}]$ in Problem (5.6) as the *preference* function of player $s \in S$ and assume that $r_s > m_s \geqslant 0$. Note that if $r_s = m_s$ then player s does not have any flexibility. So, it should be simply removed from the game. The advantage of preference over payoff is that preferences are easily comparable as they are unitless and are from the same order of magnitude. More importantly, unlike payoffs, the preference values are meaningful. Specifically, the value of one for the preference function of player $s \in S$ implies that the obtained solution is 100% similar to the player's ideal outcome that the player is looking for, which is m_s. Similarly, the value of zero for the preference function of player $s \in S$ implies that the obtained solution is 100% similar to the player's worse outcome that the player is trying to avoid, which is r_s. As an aside, we note that in Problem (5.6), it is not possible to obtain values worse than r_s for the utility of player $s \in S$ because there is a solid constraint for it. However, there is no constraint to impose that values better/smaller than m_s are not allowed for player $s \in S$. This implies that if, instead of computing m_s exactly, we simply approximate it heuristically, then there will be a chance that the preference functions take values larger than one in theory (while in practice, this will be unlikely in this study). With this in mind, in this paper, we will convert the fluency map optimization problem to Problem (5.6) and solve it to find the Nash optimal plan while m_s and r_s are both approximated heuristically for each $s \in S$ based on the outcome of the quadratic fluency map optimization problem. In this study, we do not impose constraints to bound the utility of player $s \in S$ from below by m_s because by doing so a desirable property of our approach described in Proposition 5.1 (see Section 5.4) will no longer hold.

5.2.2 Solution Approaches

Problem (5.6) is sometimes referred to as a Maximum Multiplicative Program (MMP) in the literature of optimization [4, 1, 87]. In this study, we assume that $\mathcal{X}$ is represented by only linear constraints. There are several approaches for solving MMPs, such as using nonlinear solvers or solving the log-transformation form of the problem; however, a more efficient solution approach is to transform the objective function of an MMP into second-order cone constraints using the technique introduced by [51]. In order to do so, we first observe that Problem (5.6) can be reformulated as a geometric-mean optimization as follows,

$$\max\left\{\gamma :\ 0 \leqslant \gamma \leqslant \big(\prod_{s\in S}[\frac{r_s - u_s(\boldsymbol{d})}{r_s - m_s}]^{p_s}\big)^{\frac{1}{\sum_{s\in S} p_s}},\ \boldsymbol{d} \in \mathcal{X},\ u_s(\boldsymbol{d}) \leqslant r_s\ \forall s \in S\right\},$$

where γ is a non-negative variable representing the geometric mean of the NSW function. It is evident that optimizing the reformulated problem and having $\bar{\gamma}$ as its optimal objective value is the same as optimizing Problem (5.6) whose optimal objective value will be equal to $\bar{\gamma}^{\sum_{s\in S} p_s}$. By letting k be the smallest integer value satisfying $2^k \geqslant \sum_{s\in S} p_s$ and by introducing a set of non-negative variables and constraints, the geometric mean constraint can be replaced, and the problem can be reformulated as

$$\begin{aligned}
\max \quad & \gamma \\
\text{s.t.:} \quad & 0 \leqslant \gamma \leqslant \sqrt{\tau_1^{k-1}\tau_2^{k-1}} \\
& 0 \leqslant \tau_j^l \leqslant \sqrt{\tau_{2j-1}^{l-1}\tau_{2j}^{l-1}} && \text{for } j = 1,\dots,2^{k-l} \text{ and } l = 1,\dots,k-1, \\
& 0 \leqslant \tau_j^0 = (\frac{r_s - u_s(\boldsymbol{d})}{r_s - m_s}) && \text{for } j = \big(\sum_{l=1}^{s-1} p_l\big) + 1,\dots,\big(\sum_{l=1}^{s} p_l\big) \text{ and } s = 1,\dots,|S|, \\
& 0 \leqslant \tau_j^0 = \gamma && \text{for } j = \big(\sum_{s=1}^{|S|} p_s\big) + 1,\dots,2^k, \\
& \boldsymbol{d} \in \mathcal{X}, \\
& u_s(\boldsymbol{d}) \leqslant r_s && \forall s \in S.
\end{aligned}$$

Observe that any constraint of the form $\{a, b, c \geqslant 0 : a \leqslant \sqrt{bc}\}$ is a second-order cone constraint because it is equivalent to $\{a, b, c \geqslant 0 : \sqrt{a^2 + (\frac{b-c}{2})^2} \leqslant \frac{b+c}{2}\}$. Note that any second-order cone constraint is convex. This combined with the fact that $\mathcal{X}$ is assumed to only contain linear constraints suggest that the proposed reformulation can be solved by a convex programming solver as long as $\boldsymbol{u}(\boldsymbol{d})$ consists of only convex functions. Note that in the reformulation, the only constraint that does not look convex is

$$0 \leqslant \tau_j^0 = \left(\frac{r_s - u_s(\boldsymbol{d})}{r_s - m_s}\right) \qquad \text{for } j = \Big(\sum_{l=1}^{s-1} p_l\Big) + 1, \ldots, \Big(\sum_{l=1}^{s} p_l\Big) \text{ and } s = 1, \ldots, |S|.$$

However, since the problem is in the form of maximization, the constraint can be written in the form of inequality as follows,

$$0 \leqslant \tau_j^0 \leqslant \left(\frac{r_s - u_s(\boldsymbol{d})}{r_s - m_s}\right) \qquad \text{for } j = \Big(\sum_{l=1}^{s-1} p_l\Big) + 1, \ldots, \Big(\sum_{l=1}^{s} p_l\Big) \text{ and } s = 1, \ldots, |S|.$$

This itself is equivalent to

$$\begin{aligned}
&0 \leqslant \tau_j^0 && \text{for } j = \Big(\sum_{l=1}^{s-1} p_l\Big) + 1, \ldots, \Big(\sum_{l=1}^{s} p_l\Big) \text{ and } s = 1, \ldots, |S|,\\
&0 \leqslant r_s - \tau_j^0(r_s - m_s) && \text{for } j = \Big(\sum_{l=1}^{s-1} p_l\Big) + 1, \ldots, \Big(\sum_{l=1}^{s} p_l\Big) \text{ and } s = 1, \ldots, |S|,\\
&u_s(\boldsymbol{d}) \leqslant r_s - \tau_j^0(r_s - m_s) && \text{for } j = \Big(\sum_{l=1}^{s-1} p_l\Big) + 1, \ldots, \Big(\sum_{l=1}^{s} p_l\Big) \text{ and } s = 1, \ldots, |S|.
\end{aligned}$$

The last constraint is obviously convex if $u_s(\boldsymbol{d})$ is convex. As an aside, since the linear expression $r_s - \tau_j^0(r_s - m_s)$ is non-negative, it can be replaced by a non-negative (dummy) variable for simplicity (if needed). Also, we observe that in the presence of the last constraint (and since by construction $\tau_j^0 \geqslant 0$, $r_s > m_s \geqslant 0$, and $r_s - \tau_j^0(r_s - m_s) \geqslant 0$), we can simply remove the constraint $u_s(\boldsymbol{d}) \leqslant r_s$ from the reformulation for each $s \in S$ as they will be redundant. Overall, a nice feature of employing the above-explained transformation is that it can be directly solved by powerful commercial solvers such as CPLEX and Gurobi if $\boldsymbol{u}(\boldsymbol{d})$ is a vector of linear or convex quadratic functions [3, 86]. That is why we use this transformation in this study.

5.3 Proposed Methodology

In this section, we will explain our proposed methodology for transforming a fluency map optimization problem to a bargaining game. As mentioned in Section 5.1.3, we do the transformation in organ level by considering each organ in patient's body as a player of the bargaining game, which will have $|S|$ players by letting S denote the set of all body organs (under consideration). We also categorize body voxels into different groups based on the organ that they belong to, and we let V_s represent the set of all voxels in organ $s \in S$, where $V = \bigcup_{s \in S} V_s$. Moreover, our approach requires a dose prescription and a penalty function for each organ as inputs of the problem. So, we re-define $\boldsymbol{l} := (l_1, \cdots, l_{|S|})$ and $\boldsymbol{u} := (u_1, \cdots, u_{|S|})$ as the prescription vectors where l_s and u_s show the lower and upper dose level prescribed for organ $s \in S$. As for the penalty function, we mentioned in Section 5.1.1 that our proposed methodology is not limited to a specific form of penalty functions. However, we use the quadratic form of penalty functions in this research due to its popularity.

In Problem (5.3), we provided the general form of the quadratic penalty function for fluency map optimization problems. Note that by dividing voxels into organ categories, we can re-define Problem (5.3), i.e., quadratic fluency map optimization, as

$$\min_{\boldsymbol{d} \in \mathcal{D}} \sum_{s \in S} w_s f_s(\boldsymbol{d}) = \sum_{s \in S} w_s \Big(\sum_{v \in V_s} \alpha_v [(d_v - u_s)_+^2 + (l_s - d_v)_+^2] \Big),$$

where $f_s(\boldsymbol{d})$ is the penalty function of organ $s \in S$. We also define $\boldsymbol{w} := (w_1, \cdots, w_{|S|})$ as the vector of positive weights where w_s represents the importance of organ $s \in S$. Having the dose prescription and penalty functions provided as inputs, we need to identify the four pieces of information mentioned in Section 5.2 to create the bargaining game for the fluency map optimization.

5.3.1 The Feasible Set of Actions

We define the feasible set of actions as

$$\mathcal{X} := \{\boldsymbol{d} : \boldsymbol{d} \in \mathcal{D},\ \hat{l}_s \leqslant d_v \leqslant \hat{u}_s \quad \forall v \in V_s \text{ and } \quad \forall s \in S\}, \tag{5.8}$$

where $\hat{\boldsymbol{l}} := (\hat{l}_1, \cdots, \hat{l}_{|S|})$ and $\hat{\boldsymbol{u}} := (\hat{u}_1, \cdots, \hat{u}_{|S|})$ are the vectors of bounds; $\hat{l}_s$ and $\hat{u}_s$ represent the lower and upper bound for organ $s \in S$, respectively. Observe that the feasible set of actions, $\mathcal{X}$, is precisely the set of all possible fluency maps, i.e., $\mathcal{D}$, with some additional constraints on radiation doses delivered to each voxel within each organ. Note that by replacing $\hat{\boldsymbol{l}}$ and $\hat{\boldsymbol{u}}$ with their prescribed values, i.e., $\boldsymbol{l}$ and $\boldsymbol{u}$, our feasible set of actions will be exactly the fluency maps that satisfy the prescription. However, as we mentioned in Section 5.1.1, a common problem with the clinical dose prescriptions is that they are often infeasible, i.e., $\mathcal{X} = \varnothing$. This is problematic because, in the bargaining game, each player/organ will start to negotiate from their respective references, and some of the references can be out of reach due to infeasibility. Therefore, to solve this issue, we will define $\hat{\boldsymbol{l}}$ and $\hat{\boldsymbol{u}}$ such that they are feasible and have the minimal deviation from their initial prescription.

To do so, we solve Problem (5.9), which is the quadratic fluency map optimization, to find a feasible solution for the fluency map problem.

$$\tilde{\boldsymbol{d}} \in \underset{\boldsymbol{d} \in \mathcal{D}}{\arg\min} \Big\{ \sum_{s \in S} w_s f_s(\boldsymbol{d}) \Big\} \tag{5.9}$$

Next, we define $\hat{l}_s$ and $\hat{u}_s$ as the minimum and the maximum doses delivered to the voxels of organ $s \in S$ as shown in Equation (5.10),

$$\hat{l}_s = \min_{v \in V_s} \{\tilde{d}_v\} \quad ; \quad \hat{u}_s = \max_{v \in V_s} \{\tilde{d}_v\}. \tag{5.10}$$

Note that, as mentioned earlier, the values defined for $\hat{\boldsymbol{l}}$ and $\hat{\boldsymbol{u}}$ using the proposed method are feasible and closest to the prescribed values.

Remark 5.1. The vector of weights, i.e., $\boldsymbol{w} := (w_1, \cdots, w_{|S|})$, in Problem (5.9) can be manipulated to get different $\tilde{\boldsymbol{d}}$ and consequently different $\hat{\boldsymbol{l}}$ and $\hat{\boldsymbol{u}}$. Therefore, if a tighter/looser bound for a specific organ is favorable, one can increase/decrease the organ's weight.

We note that, if the initially prescribed dose levels are certainly feasible, one can use the prescribed dose levels to define the feasible set of actions as shown in

$$\mathcal{X} := \{\boldsymbol{d} : \boldsymbol{d} \in \mathcal{D},\ l_s \leqslant d_v \leqslant u_s \quad \forall v \in V_s \text{ and } \quad \forall s \in S\},$$

and use the ideal dose levels, i.e.,

$$l_s = u_s = \begin{cases} 0 & \text{when } s \text{ is OAR} \\ T_s & \text{when } s \text{ is PTV} \end{cases},$$

to define the penalty functions. By so doing, the purpose of the bargaining game will be to find a fluency map that is closest to the ideal dose levels but does not violate the prescription.

5.3.2 Utility Functions

The utility function of organ $s \in S$ is basically its input penalty function, i.e., $f_s(\boldsymbol{d})$, which is to be minimized. By means of clarity, since we are using quadratic penalty functions in this research, the utility function of organ $s \in S$ is

$$u_s(\boldsymbol{d}) = f_s(\boldsymbol{d}) = \sum_{v \in V_s} \alpha_v[(d_v - u_s)_+^2 + (l_s - d_v)_+^2], \tag{5.11}$$

where α_v is the importance weight of voxel v in organ s, d_v is the total dose delivered to voxel v in organ s, and l_s and u_s are the lower and upper dose levels prescribed for organ s, respectively. The point of considering the penalty functions as our utility functions is that we are designing a bargaining game to find a fluency map that minimizes the deviation from the prescribed dose levels but does not violate the feasible dose levels defined in Section 5.3.1.

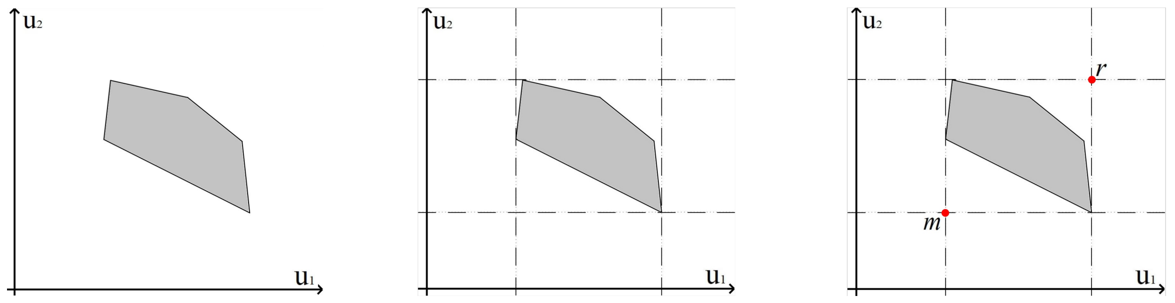
In other words, we are trying to find a fluency map that minimizes the utility/penalty functions of players/organs shown in Equation (5.11) while satisfying the following constraint,

$$\sum_{v \in V_s} \alpha_v[(d_v - \hat{u}_s)_+^2 + (\hat{l}_s - d_v)_+^2] = 0 \qquad \forall s \in S \ .$$

5.3.3 Disagreement Point

In this section, we explain how the payoff of players/organs in the disagreement point can be calculated. The importance of disagreement point is that, in a bargaining game, players/organs will assess their utility/penalty functions with respect to those of other players/organs and will consider a minimum and a maximum expected outcome for themselves. Then, they will not accept any solution resulting in less than their minimum expectation and will try to obtain an outcome as close to their maximum expectation as possible.

By means of illustration, we provide an example of a bargaining game with two players ($|S| = 2$) trying to minimize their utility functions. Figure 5.1a, shows the image of the feasible set of actions in the payoff space, i.e., the feasible values that the utility functions of players can achieve. In Figure 5.1b, the dashed lines show the maximum and the minimum values of the utility functions. Since both players try to minimize their respective utility functions, the ideal outcome of the game is when both players achieve their minimum feasible utility, point $\boldsymbol{m}$ in Figure 5.1c, and the worst outcome of the game is when both players achieve their maximum feasible utility, point $\boldsymbol{r}$ in Figure 5.1c. In other words, point r is the disagreement point of the game, and the players will not accept any solution worse than it. In addition, point m is the ideal point of the game, and the goal of the coalition is to find a final solution that is closest to it.



(a) feasible region (b) boundries (c) ideal and worst points

Figure 5.1: An illustration of the feasible set of a bargaining game with two players ($|S| = 2$) in the payoff space

The process of defining the disagreement point for two-player games ($|S| = 2$) is relatively easy, however as the number of players increases, this process becomes more challenging and computationally expensive [88, 89]. Therefore, we simply propose an approximation technique to compute the disagreement point for fluency map problems.

In the proposed technique, we first need to solve a pseudo-lexicographic optimization operation for each player/organ $s \in S$. Note that a normal lexicographic optimization operation consists of two objective functions: a primary one and a secondary one. The operation seeks to optimize the secondary objective function over all optimal solutions of the primary objective function. The proposed pseudo-lexicographic optimization works similarly but it simply avoids quadratic terms to keep the operation computationally manageable. With this in mind, the operation starts by first solving

$$\bar{\boldsymbol{d}}^s \in \arg\min_{\boldsymbol{d} \in \mathcal{X}} \big\{ u_s(\boldsymbol{d}) \big\} \tag{5.12}$$

and then solving

$$\bar{\bar{\boldsymbol{d}}}^s \in \arg\min_{\boldsymbol{d} \in \mathcal{X}} \Big\{ \sum_{s' \in S \setminus \{s\}} u_{s'}(\boldsymbol{d}) : \ \bar{d}^s_v - \epsilon \leqslant d_v \leqslant \bar{d}^s_v + \epsilon \quad \forall v \in V_s \Big\}. \tag{5.13}$$

Observe that, for each player/organ $s \in S$, two optimization problems need to be solved. In the first problem, the player's/organ's utility/penalty function will be minimized on the feasible set of actions defined in Section 5.3.1. This optimization problem finds the ideal utility/penalty value that the player/organ s can achieve, which is $u_s(\bar{\boldsymbol{d}}^s)$. Then, in the second optimization problem, the sum of all other players'/organs' utility/penalty functions will be minimized on the feasible set of actions with some additional constraints. The additional constraints are basically assuring that

$$u_s(\boldsymbol{d}) = u_s(\bar{\boldsymbol{d}}^s).$$

However, such constraints will add many quadratic terms to the model. So instead, we use the following set of constraints,

$$\bar{d}_v^s - \epsilon \leqslant d_v \leqslant \bar{d}_v^s + \epsilon \qquad\qquad \forall v \in V_s \ ,$$

where ϵ is a sufficiently small positive constant to avoid numerical issues.

By solving the pseudo-lexicographic problem for all players/organs, the ideal expected outcome of each player $s \in S$ is

$$m_s = u_s(\bar{\boldsymbol{d}}^s).$$

Therefore, we refer to $\bar{\boldsymbol{d}}^s$ as the ideal fluency map for organ s. Note that the ideal fluency map of one organ is not necessarily ideal for other organs since when computing the ideal fluency map for one organ, the other organs are not considered. The worst expected outcome of player $s \in S$ can be estimated by

$$r_s = u_s(\ddot{\boldsymbol{d}}^s),$$

where

$$\ddot{\boldsymbol{d}}^s \in \arg\max_{s' \in S} \left\{ u_s(\bar{\bar{\boldsymbol{d}}}^{s'}) \right\}.$$

Therefore, we refer to $\ddot{\boldsymbol{d}}^s$ as the worst fluency map for organ s. Note that again the worst fluency map of one organ is not necessarily worst for other organs.

5.3.4 Negotiation Powers

The vector of negotiation powers, $\boldsymbol{p}$, provides the practitioners with the ability to further modify the final fluency map by putting more emphasis on an organ by increasing its negotiation power. In other words, having defined the feasible set of actions, utility functions, and the disagreement point, the negotiation powers can help improve the final plan. As mentioned in Section 5.1.3, negotiation powers are control levers for the fluency map that, to the best of our knowledge, are introduced in this research for the first time. That said, by developing an update role for negotiation powers, an automated algorithm can be constructed to improve the final plan. However, this does not fit in the scope of the current research as we are mainly focusing on introducing the concept of bargaining games to fluency map optimizations. Therefore, we leave it as a future research guideline.

Remark 5.2. The negotiation powers are different from the organs' weights, i.e., $\boldsymbol{w}$. In particular, the vector $\boldsymbol{w}$ emphasizes the importance of different players/organs in finding the feasible dose levels, where a higher weight results in a tighter dose level. However, the vector $\boldsymbol{p}$ emphasizes the power of player/organ in the bargaining process, where a higher power results in a final solution more preferable for the player/organ.

5.3.5 Fluency Map Bargaining Game

In Sections 5.3.1 - 5.3.4, we explained our proposed methodology to define the necessary pieces of information to create a bargaining game. Now, we formally define the fluency map bargaining game based on Problem (5.6) as the following optimization problem,

$$
\begin{aligned}
\max \quad & \prod_{s\in S} \left[\frac{r_s - u_s(\boldsymbol{d})}{r_s - m_s}\right]^{p_s} & & \\
\text{s.t.} \quad & u_s(\boldsymbol{d}) \leqslant r_s & \forall s \in S, & \\
& d_v = \sum_{n\in N} D_{vn} x_n & \forall v \in V, & \qquad (5.14)\\
& \hat{l}_v \leqslant d_v \leqslant \hat{u}_v & \forall v \in V, & \\
& x_n \geqslant 0 & \forall n \in N. &
\end{aligned}
$$

To find the Nash optimal solution of the fluency map bargaining game, practitioners can use the approach explained in Section 5.2.2 to convert the objective function of Problem (5.14) into second-order cone constraints and then solving the reformulation using commercial solvers such as CPLEX and Gurobi. In terms of implementation, we note that as mentioned in Section 5.2.2, for each $s \in S$, when applying the transformation the constraint $u_s(\boldsymbol{d}) \leqslant r_s$ will become redundant and can be removed. Additionally, we know that by construction, when computing m_s and r_s where $s \in S$, the constraint $\hat{l}_v \leqslant d_v \leqslant \hat{u}_v$ is considered for each $v \in V$. This combined with the fact that $u_s(\boldsymbol{d}) \leqslant r_s$ for each $s \in S$ suggest that when solving the optimization problem, most likely the constraint $\hat{l}_v \leqslant d_v \leqslant \hat{u}_v$ will be naturally satisfied for each $v \in V$. This is because the optimization problem attempts to make $u_s(\boldsymbol{d})$ closer to m_s for each $s \in S$. Therefore, during the course of this research, we found out that it is computationally significantly better to remove the constraint $\hat{l}_v \leqslant d_v \leqslant \hat{u}_v$ for each $v \in V$ when solving the fluency map bargaining game. Note that by removing these constraints, m_s is no longer a true lower bound for the utility of organ $s \in S$, i.e., it will become an approximate bound. So, in that case, based on our discussions in Section 5.2, it is possible that preference functions take values larger than one in theory, however that would be unlikely in practice. Removing these constraints also has an important theoretical advantage that will be discussed in the next section, see Proposition 5.1.

5.4 Theoretical Discussion

In this section, we explore our proposed fluency map bargaining game (in Section 5.3.5) from the theoretical perspective. To do so, we first introduce some concepts from the field of multi-criteria optimization as it is known that radiotherapy treatment planning has a strong connection with this field (see for instance [90, 91, 92, 93, 94, 95]). Specifically, it is known that most existing methods for the fluency map optimization problem can be viewed as different approaches for solving the following multi-criteria optimization problem,

$$\min_{\boldsymbol{d} \in \mathcal{D}} \{f_1(\boldsymbol{d}), \cdots, f_{|S|}(\boldsymbol{d})\}.$$

Recall that $f_s(\boldsymbol{d})$ is the penalty/objective function defined for organ $s \in S$. Unfortunately, due to the conflicting nature of the objective functions, it is often impossible to find a feasible solution, i.e., fluency map, that can minimize all penalty functions simultaneously. Hence, the goal of many existing state-of-the-art techniques, when solving the fluency map optimization problem, is to find a *Pareto-optimal* solution (see Definition 5.1) that can desirably balance the conflicts between the objectives.

Definition 5.1. A feasible fluency map, $\widehat{\boldsymbol{d}} \in \mathcal{D}$, is Pareto-optimal if there exists no other feasible fluency map, $\check{\boldsymbol{d}} \in \mathcal{D}$, such that

$$\begin{aligned} & f_s(\check{\boldsymbol{d}}) \leqslant f_s(\widehat{\boldsymbol{d}}) && \forall s \in S, \\ & f_s(\check{\boldsymbol{d}}) < f_s(\widehat{\boldsymbol{d}}) && \exists s \in S. \end{aligned}$$

In the literature of radiotherapy, there are two main categories of solution methods for computing a desirable Pareto-optimal solution for the fluency map problem [95]. We refer to theses two categorizes as *non-automated* and *automated* in this paper, and the main difference between them is that the level of interactions with decision maker(s) is more significant in the non-automated approaches. Specifically, in the non-automated approaches, the focus is on generating the entire set or a large subset of Pareto-optimal solutions in order to enable decision makers to navigate through them and identify a desirable solution. As an aside, we note that some existing commercial treatment planning software, e.g., RayStation[5.1], are developed based on the non-automated approaches. Although the non-automated approaches are a valuable source for understanding the trade-offs between the conflicting objectives, they have two main disadvantages. First, computing the entire set (or a large subset) of Pareto-optimal solutions can be computationally expensive, i.e., time consuming. The second disadvantage is that it is long argued in the field of multi-criteria optimization that presenting too many Pareto-optimal solutions to decision makers can sometimes confuse the decision makers and make selecting a preferred solution difficult [7, 8]. So, the process of selecting a desirable Pareto-optimal solution itself can be burdensome as well.

To overcome the above-mentioned challenges, the automated solution approaches seek to directly return a desirable Pareto-optimal solution based on the decision maker's *wish-list*, which is a solution-approach and decision-maker dependent list including information such as the priority of each objective, their corresponding acceptable values, etc. In other words, in the context of the automated solution approaches, a desirable Pareto-optimal solution can be viewed as the closest solution to the wish-list of decision makers (rather than a solution that decision makers select by inspection). Since there is no unique and trivial way to measure the closeness, identifying the closest solution is not a trivial task too. Consequently, not surprisingly, several automated solution approaches exist in the literature. In fact all the solution approaches that we discussed through this paper, e.g., the quadratic fluency map optimization or our proposed fluency map bargaining game, belong to the category of automated solution approaches. Two other approaches in the same category are the so-called lexicographic optimization method and the 2-phase ϵ-constraint method developed by [96]; We will later use these two methods in our computational study.

[5.1] https://www.raysearchlabs.com/multi-criteria-optimization-treatment-planning/

We note that all automated solution approaches seek to minimize the level of interactions with decision makers in order to speed up the entire planning process. However, if decision makers are not happy with the plan generated by an automated solution approach, they still have the option to make changes to their wish-lists in order to force the approach to return a possibly different solution. Of course, the hope of all automated solution approaches is to avoid such an interactive process (or minimize the number of iterations). In the context of our approach, the wish-list can be viewed as the negotiation powers, weights, etc. With this in mind, we next prove that our approach is indeed an automated solution approach. In order to do so, it suffices to prove that the proposed approach always returns a Pareto-optimal solution. We then explain that our approach comes with an additional natural advantage which makes its outcome likely to be acceptable for decision makers.

Proposition 5.1. *The proposed fluency map bargaining game, i.e., Problem* (5.14)*, always returns a Pareto-optimal solution.*

Proof. Recall that in the context of our research, $u_s(\boldsymbol{d}) = f_s(\boldsymbol{d})$ for each $s \in S$, see Section 5.3.2. Moreover, based on our discussions in Section 5.3.5, the constraint $\hat{l}_v \leqslant d_v \leqslant \hat{u}_v$ should be removed from Problem (5.14) for each $v \in V$. So, the proposed fluency map bargaining game can be stated as

$$\begin{array}{lll} \max & \prod_{s \in S} [\frac{r_s - f_s(\boldsymbol{d})}{r_s - m_s}]^{p_s} & \\ \text{s.t.} & f_s(\boldsymbol{d}) \leqslant r_s & \forall s \in S, \\ & \boldsymbol{d} \in \mathcal{D}. & \end{array}$$

We now prove the assertion by contradiction. Let $\boldsymbol{d}^*$ be an optimal solution of the above-mentioned problem and suppose that it is not Pareto-optimal. In that case, following Definition 5.1, there must exist a solution, $\check{\boldsymbol{d}} \in \mathcal{D}$, that *dominates* $\boldsymbol{d}^*$, i.e.,

$$\begin{aligned} & f_s(\check{\boldsymbol{d}}) \leqslant f_s(\boldsymbol{d}^*) && \forall s \in S, \\ & f_s(\check{\boldsymbol{d}}) < f_s(\boldsymbol{d}^*) && \exists s \in S. \end{aligned}$$

First note that since $f_s(\boldsymbol{d}^*) \leqslant r_s$ for all $s \in S$, we must have that $f_s(\check{\boldsymbol{d}}) \leqslant r_s$ for all $s \in S$. This is because otherwise (by definition) there must exist $s \in S$ such that $f_s(\boldsymbol{d}^*) \leqslant r_s < f_s(\check{\boldsymbol{d}})$ which clearly violates the assumption that $\check{\boldsymbol{d}}$ dominates $\boldsymbol{d}^*$. This combined with the fact that $\check{\boldsymbol{d}} \in \mathcal{D}$ imply that $\check{\boldsymbol{d}}$ is a feasible solution for the proposed fluency map bargaining game. Now, since by assumptions $r_s > m_s \geqslant 0$ and $p_s > 0$ for all $s \in S$, we have that

$$\prod_{s \in S} [\frac{r_s - f_s(\boldsymbol{d}^*)}{r_s - m_s}]^{p_s} \quad < \quad \prod_{s \in S} [\frac{r_s - f_s(\check{\boldsymbol{d}})}{r_s - m_s}]^{p_s}.$$

Consequently, $\boldsymbol{d}^*$ cannot be an optimal solution as $\check{\boldsymbol{d}}$ has a better objective value for the fluency map bargaining game (a contradiction). $\square$

Following Proposition 5.1, as long as the assumption $r_s > m_s \geqslant 0$ and $p_s > 0$ for all $s \in S$ hold, the returned solution by solving the proposed fluency map bargaining game is definitely Pareto-optimal. So, in essence, users have the full flexibility to choose any arbitrary values for those parameters in a clinical setting. The only consideration is that due to the existence of the constraint $f_s(\boldsymbol{d}) \leqslant r_s$ for all $s \in S$ in the fluency map bargaining game, there is a possibility that the fluency map bargaining game becomes infeasible for some arbitrary choices of the disagreement point. That is the main reason that in our proposed approach, we solve $2|S| + 1$ optimization problems to compute the disagreement points such that the fluency map bargaining game remains feasible.

In addition to the Pareto-optimality, our proposed approach comes with a unique desirable property which is balancing *efficiency* and *fairness* [97, 12, 85]. Intuitively, efficiency means maximizing the total size of the *cake* while fairness means dividing/sharing the cake as equitable as possible among players with respect to their negotiation powers, i.e, those with higher negotiation powers are likely to get a larger proportion of the cake. In the context of our research, the 'cake' refers to any feasible treatment plan; The share of a player/organ from the cake refers to the similarity ratio of the treatment plan to its ideal plan; Finally, the size of the cake is the sum of all similarity ratios.

In a perfect world, each organ will receive a radiation plan which is 100% similar to its true ideal plan. However, that may not be achievable in practice, and therefore, there can be trade-offs between fairness and efficiency. Given the disagreement point and negotiation powers of players, it is known in the field of game theory that a natural way of balancing efficiency and fairness is to maximize the NSW function over the feasible set of actions [97, 86]. In other words, by maximizing the NSW function, we are implicitly trying to find a solution/plan that maximizes the size of the cake. If by so doing, it turns out that the cake is divided completely fairly then that solution has to be optimal. Otherwise, the approach implicitly attempts to give up on the size of the cake in order to improve fairness. Overall, balancing efficiency and fairness is an important property of our approach as it suggests that our proposed approach naturally attempts to avoid generating extreme Pareto-optimal solutions/plans which are significantly biased towards some players (unless their negotiation powers are considerably higher). This is desirable as such extreme plans are typically unacceptable for decisions makers [92].

5.5 Numerical Results

In this section, we conduct a numerical study by implementing the proposed bargaining game on the TG-119 and the liver case provided in the CORT dataset [69]. For each case study, we compare our results with a state-of-the-art solution approach, i.e., quadratic fluency map optimization, in this section. Note that the quadratic fluency map optimization is precisely Problem (5.9). In our numerical study, we only consider the weights in organ level, i.e., $\boldsymbol{w}$, and negotiation powers, i.e., $\boldsymbol{p}$. Therefore, the voxel-level importance weights are not considered, i.e., we assume that $\alpha_v = 1$ for each $v \in V$. For both case studies, we followed the ideal dose prescription of the form

$$l_s = u_s = \begin{cases} 0 & \text{when } s \text{ is OAR} \\ T_s & \text{when } s \text{ is PTV} \end{cases},$$

where OAR includes all healthy organs of the body and PTV are the cancerous organs to be eliminated. Note that the organ weights, $\boldsymbol{w}$, play an important role in defining the feasible set of actions of the bargaining game. Therefore to conduct a more comprehensive numerical study, we considered two different cases for them. Specifically, for the TG-119 case, organ weights are selected such that they prioritize OAR sparing over PTV coverage. However, for the liver case, the organ weights are selected to prioritize PTV coverage over OAR sparing. We implemented our methodology in C++ and used CPLEX 12.9 as our solver. 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 the default settings of CPLEX.

5.5.1 TG-119 Case

For TG-119 case, we used all the five beam angles provided in the CORT dataset [69]. The case includes a total of 418 beamlets, a total of 599,440 body voxels, and three body organs. The organs are Target with 7,429 voxels, Core with 1,280 voxels, and Normal tissues with 302,953 voxels. Note that the remainder of 287,778 body voxels receive a maximum of zero radiation, and therefore, we did not include them in the model and in the DVH. We assumed a prescription dose of 55 Gray (Gy) for the Target. To prioritize Core sparing over PTV coverage, and PTV coverage over body sparing, we simply considered organ weights relative to the organs' sizes as follows,

$$w_C = \frac{1}{1,280} \quad , \quad w_T = \frac{1}{7,429} \quad , \quad w_N = \frac{1}{287,778},$$

where w_c, w_T, and w_N are the importance weights of Core, Target, and Normal tissues, respectively. Following Remark 5.1, these bounds result in the tightest dose levels for Core and the loosest dose levels for Normal tissues. We followed the steps in Section 5.3.1 to define the feasible set of actions and to calculate the feasible dose levels for each player/organ, which are provided in Table 5.1.

Table 5.1: TG119 case: Feasible dose levels

	$\hat{l}$	$\hat{u}$
Core	0.56	15.09
Target	31.25	64.46
Normal	0.0	55.90

Following Section 5.3.3 to define the disagreement point, we calculated the ideal and worst expected utility/penalty values for each player/organ. Figure 5.2 depicts the DVH of the ideal (solid line) and the worst (dashed line) fluency maps of each player/organ. Note that for each organ $s \in S$, its ideal DVH can be generated based on $\bar{\boldsymbol{d}}^s$ as it corresponds to m_s (which is its expected ideal outcome) and also its worst DVH can be generated based on $\ddot{\boldsymbol{d}}^s$ as it corresponds to r_s (which is its expected worst outcome). Note that in Figure 5.2, the area between the ideal and the worst DVH of each player/organ provides the range where the solution of our Nash Social Welfare optimization is expected to lie in.

Figure 5.2 helps to have an overview of the outcome of the bargaining game. As mentioned in Remark 5.2, the negotiation powers of players/organs determine their similarity to their ideal expected outcome. Therefore, for example, by using a sufficiently large negotiation power for Target compared to the powers of Core and Normal tissues, one can obtain a Target DVH identical to its solid line in Figure 5.2. However, in such a case, a DVH similar to the dashed lines should be expected for Core and Normal tissues. By means

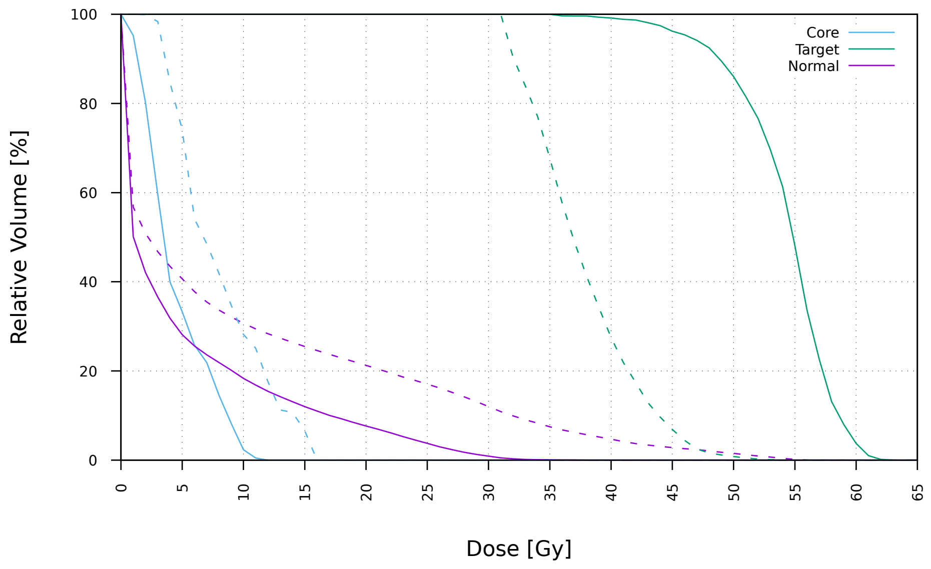


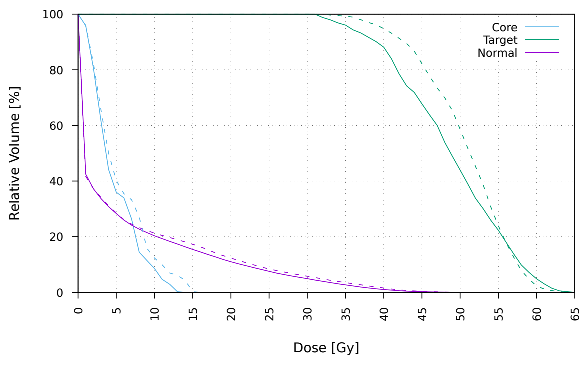
Figure 5.2: TG-119 case: DVH of the ideal (solid curves) and worst (dashed curves) utility/penalty values

of clarification, we modeled the bargaining game testing different negotiation powers. The powers we tested are

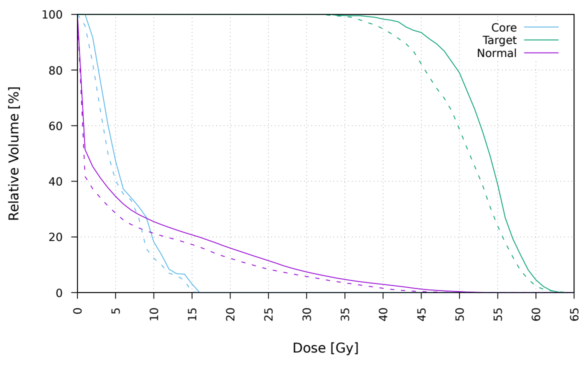
$$(p_T, p_C, p_N) \in \{(1,1,1),(21,1,1),(42,1,1),(42,10,1)\},$$

where the highest power is related to Target and is approximately equal to the ratio of the total number of body voxels divided by number of Target voxels. Specifically, we chose these powers to show the changes in the outcome of the game as the negotiation power of Target gradually increases. To compare the result of our approach with the quadratic fluency map optimization, in Figure 5.3, we provided the DVH generated by employing the tested negotiation powers in bargaining game as solid lines and the DVH of the quadratic fluency map optimization, i.e., Problem (5.9), as dashed lines.

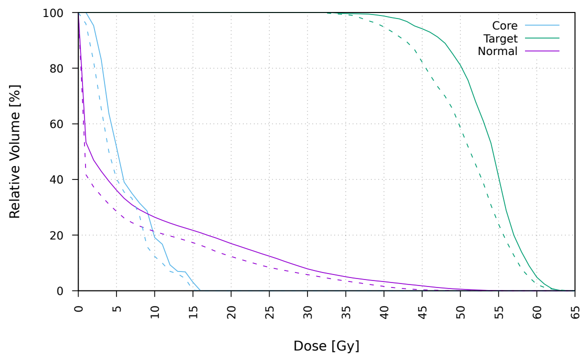
Figure 5.3a shows the DVH of the game where all players/organs have unit powers. The outcome of the bargaining problem (solid lines) shows 75%, 90.11%, and 74.9% similarity of Target, Core, and Normal to their ideal expected utility/penalty values, respectively. Note that these are the values of preference functions explained in Section 5.2.1 (that are multiplied



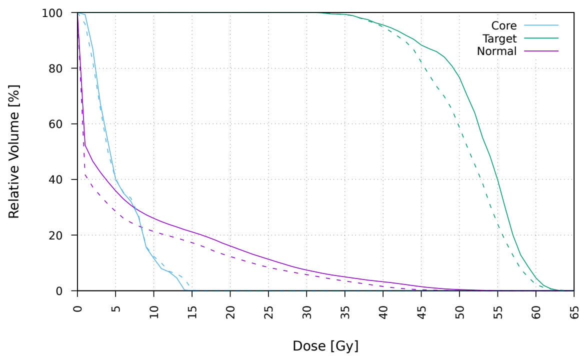
(a) $(p_T, p_C, p_N) = (1, 1, 1)$



(b) $(p_T, p_C, p_N) = (21, 1, 1)$



(c) $(p_T, p_C, p_N) = (42, 1, 1)$



(d) $(p_T, p_C, p_N) = (42, 10, 1)$

Figure 5.3: TG-119 case: DVH comparison between the Nash optimal solution (solid curves) and the quadratic optimal solution (dashed curves)

by 100) after solving the bargaining problem. The similarity percentages of the outcome of the fluency map optimization problem (dashed lines) can be also obtained by plugging its solution into the preference functions explained in Section 5.2.1. By doing so, the percentages will be 88.7%, 70.4%, and 66.5% for Target, Core, and Normal respectively. This indicates that in the bargaining process, the preference of Target is sacrificed to obtain the maximum NSW.

Figure 5.3b and Figure 5.3c show the DVH of the game where the negotiation power of Target is increased to 21 and 42, and Core and Normal have unit powers. These games resulted in 97.01% and 97.93% similarity of Target to its ideal case. The similarity percentages of Core are 55.8% and 48.92%, and of Normal are 41.7% and 34.32%. Observe that, as we increase the negotiation power of Target in comparison to the power of other players/organs, the preference value of Target increases, and it tends to obtain a DVH closer to its ideal DVH. However, the preference of other players/organs decreases, and their DVH moves closer to their worst case.

Figure 5.3d shows the case where Target has a power of 42, Core has a power of 10, and Normal has unit power. In this case, the similarity of Target, Core, and Normal to their ideal case are 93.6%, 73.3%, and 39.2%, respectively. It is clear that, due to the negotiation powers, the preference of Normal is sacrificed to ensure a higher preference for Target and then Core. Similarly, we observe from Figure 5.3d that the DVH of Core and Target has changed for the better by sacrificing the DVH of Normal compared to their DVH's generated by the quadratic fluency map optimization.

5.5.2 Liver Case

In this section, we implement our methodology on the liver case provided in the CORT dataset. In this dataset, a total of 56 beam angles are provided, of which we only use 8 that are reported as the result of the beam angle optimization algorithm developed by [77]. These 8 beam angles include a total of 519 beamlets. The body organs that we include in our model are PTV with 6,954 voxels, Skin with 465,093 voxels, Heart with 28,867 voxels, Left Kidney (KidneyL) with 1,295 voxels, Right Kidney (KidneyR) with 692 voxels, Large Bowel with 133 voxels, Liver with 52,999 voxels, Spinal Cord with 685 voxels, and Stomach with 7,789 voxels. Therefore, the total number of body voxels are equal to 564,507, all of which can receive a maximum dose of greater than zero by the included beam angles.

Since the liver case is more realistic compared to TG-119, we divide our numerical analysis in this section into two parts. In the first part, we provide an overview of the performance of the proposed algorithm compared to the quadratic optimization similar to what we did for the TG-119 case. In the second part, we provide a performance comparison between our proposed algorithm and two existing automated (multi-criteria) solution approaches including lexicographic optimization method (Lexico) and the 2-phase ϵ-constraint method (2pϵc). Our specific implementation of Lexico and 2pϵc on the liver case is provided in Appendix D1, and the interested readers can refer to [96] for the generic form of these approaches.

5.5.2.1 Overall Performance

Similar to the TG-119 case, we assume a prescription dose of 55 Gy for the PTV and zero Gy for other organs, and unlike to the TG-119 case, we looked for the feasible bounds that prioritize PTV coverage over OAR sparing. To do so, we employed the following importance weights,

$$w_s = \begin{cases} \dfrac{6,954}{564,507} & s = PTV \\ \\ \dfrac{1}{564,507} & \text{otherwise} \end{cases},$$

where w_{PTV} is the ratio of the PTV size to the total number of voxels, and other weights are the ratio of one over the total number of voxels. Then, we defined the feasible dose levels of each player/organ following Section 5.3.1, which are provided in Table 5.2. Further, in the process of defining the disagreement point, we extracted the DVH of the ideal and worst expected utility/penalty values for each player/organ, which are provided in Figure 5.4.

Table 5.2: Liver case: Feasible dose levels

	$\hat{l}$	$\hat{u}$
PTV	31.31	59.73
Skin	0.0	58.24
Heart	0.0	50.47
KidneyL	0.0	0.03
KidneyR	0.0	0.06
Large Bowl	0.0	0.02
Liver	0.0	55.14
Spinal Cord	0.0	12.14
Stomach	0.0	15.15

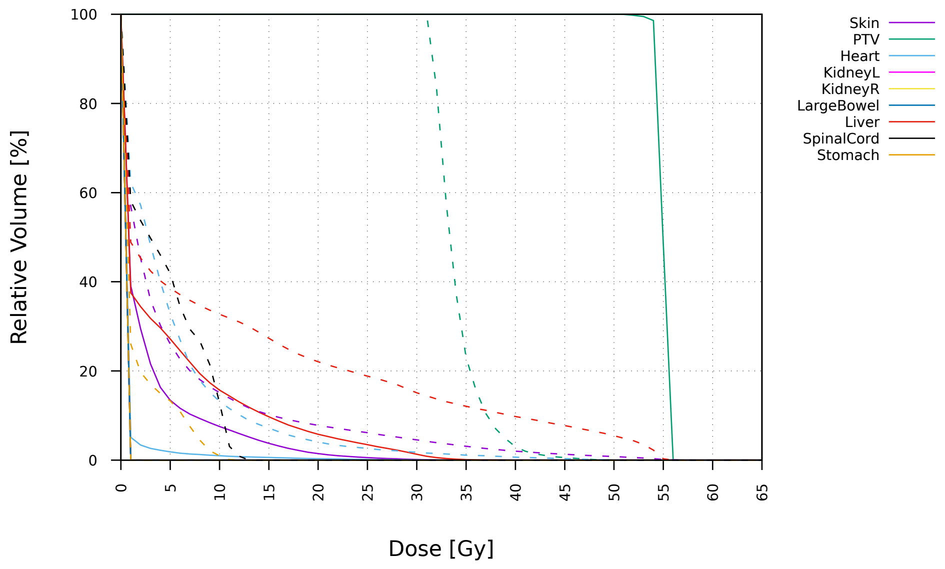
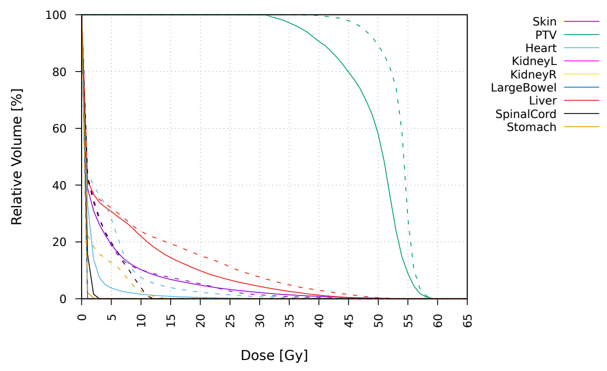


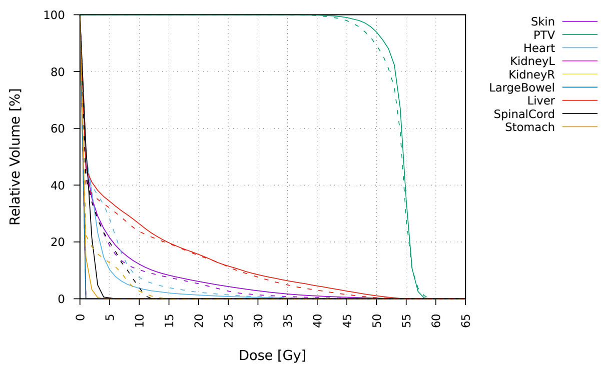
Figure 5.4: Liver case: DVH of ideal (solid curves) and worst (dashed curves) utility/penalty values

Similar to the previous section, we modeled the bargaining game using different negotiation powers, where the highest power is related to PTV which is approximately equal to the ratio of the total number of body voxels to the number of PTV voxels. Figure 5.5 compares the DVH of the Nash optimal solutions with the tested negotiation powers (solid lines) with the DVH of the quadratic fluency map optimization (dashed lines). Figure 5.5a shows the DVH of the game where all players/organs have the same unit powers. From this figure, we observe that, similar to the TG-119 case, the preference of PTV is sacrificed to obtain the maximum NSW. Moreover, the DVH of Skin has changed a little compared to the quadratic optimal solution, but the DVH of all the other players/organs have improved.

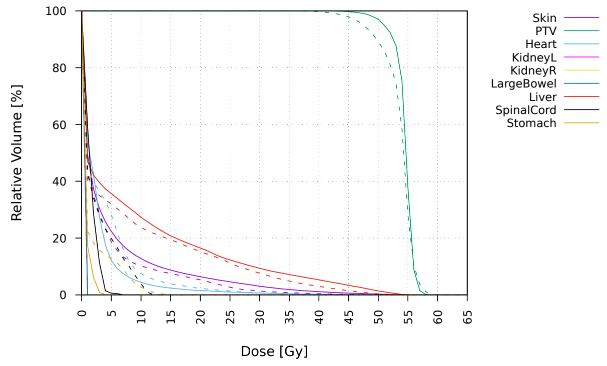
Figures 5.5b and 5.5c show the DVH of the Nash optimal solution where PTV has a negotiation power equal to 40 and 81, and the other players/organs have powers equal to one. Observe that, as the negotiation power of PTV increases, its DVH tends to move closer to its ideal case. During this change, although the quadratic fluency map optimization resulted in a better DVH for Skin and Liver, the Nash optimal solution resulted in highly improved DVH's for all other organs. Figure 5.5d shows the DVH of the Nash optimal solution where PTV has a negotiation power equal to 81, Liver has a power equal to 10, and the rest of the



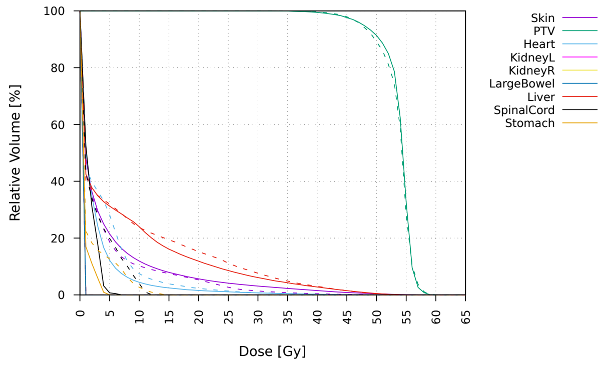
(a) $p_s = 1$ for all organs



(b) $p_{\text{PTV}} = 40$, $p_s = 1$ for others



(c) $p_{\text{PTV}} = 81$, $p_s = 1$ for others



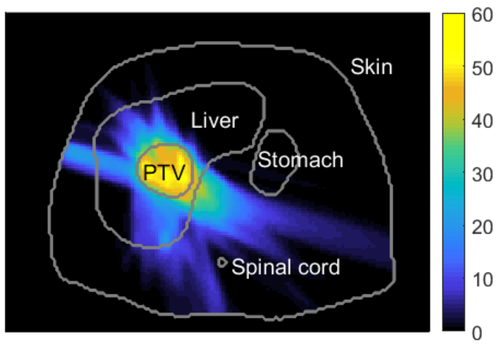
(d) $p_{\text{PTV}} = 81$, $p_{\text{Liver}} = 10$, $p_s = 1$ for others

Figure 5.5: Liver case: DVH comparison between the Nash optimal solution (solid curves) and the quadratic optimal solution (dashed curves)

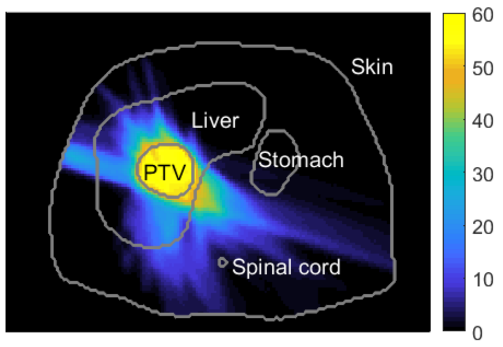
players/organs have unit powers. Observe from Figure 5.5d that the bargaining game has improved the DVH of all players/organs compared to the quadratic optimal solution with the cost of slightly sacrificing the DVH of Skin. Finally, since the liver case is more realistic (compared to TG-119), for interested readers, we report a 2D dose distribution map of its Nash optimal solution in Figure 5.6 for all four scenarios of negotiation powers.

5.5.2.2 Performance Comparison

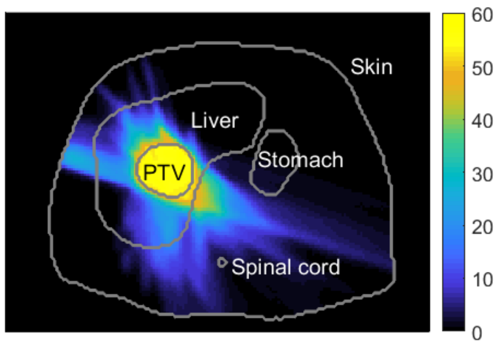
We now compare the performance of our approach with the Lexico and 2pϵc. The wish-list that we use for the Lexico and 2pϵc is summarized in Table 5.3. We note that the wish-list is designed based on the outcome of our proposed approach in Figure 5.5d. In other words, we define the wish-list in such a way that we can compare the performance of our



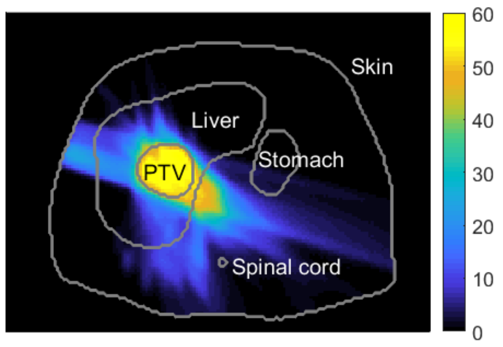
(a) $p_s = 1$ for all organs



(b) $p_{\text{PTV}} = 40$, $p_s = 1$ for others



(c) $p_{\text{PTV}} = 81$, $p_s = 1$ for others



(d) $p_{\text{PTV}} = 81$, $p_{\text{Liver}} = 10$, $p_s = 1$ for others

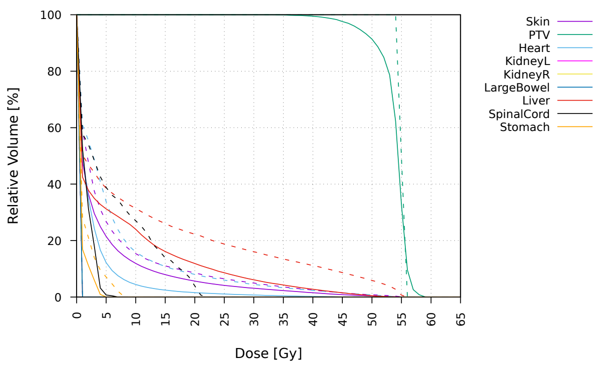
Figure 5.6: Liver case: dose distribution map of the Nash optimal solution

approach shown in Figure 5.5d with the Lexico and 2pϵc. Since in Figure 5.5d, the PTV has the highest negotiation power, we give the first priority to the PTV in the Lexico and 2pϵc. Moreover, since the liver has the second highest negotiation power in Figure 5.5d, we give the second highest priority to the liver. Finally, since all other organs have the same negotiation powers in Figure 5.5d, we give the third priority to all other organs together. In Figure 5.5d, the (quadratic) penalty value for the PTV is 68,386.38, so we set 68,386.38 as the goal for the PTV. Similarly, the (quadratic) penalty value for the liver is 8,750,124.63 in Figure 5.5d, so we set 8,750,124.63 as the goal for the liver. Finally, the sum of all (quadratic) penalty values for other organs is 42,012,837.22 in Figure 5.5d, so we set 42,012,837.22 as the goal for the all other organs together.

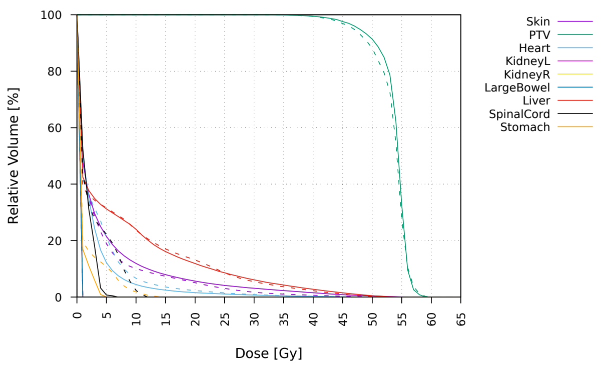
Table 5.3: Liver case: the wish-list used for the Lexico and 2pϵc.

Priority	Volume	Goal
1	PTV	68,386.38
2	Liver	8,750,124.63
3	Other organs	42,012,837.22

As an aside, we note that CPLEX 12.9 was struggling when solving the Lexico and 2pϵc due to the side constraints that these methods involve. Therefore, in order to employ Lexico and 2pϵc, we ended up using Gurobi 9.1 which is another powerful commercial solver. Figure 5.7 compares the DVH of the outcomes of the Lexico and 2pϵc (dashed lines) with the outcome of the proposed bargaining game (solid lines). From Figure 5.7a, we observe that the Lexico has resulted in an extreme solution that is mostly preferable to the PTV. Observe that the DVH of PTV in this figure is almost the same as its expected ideal DVH in Figure 5.4 while other organs are receiving doses close to their worst expected DVH. In fact, some organs (e.g., the spinal cord) are receiving DVH which is worse than our worst expected DVH. Note that this is possible because our worst expected DVH is defined based on our proposed Nash bargaining approach, i.e., the disagreement point. Therefore, since the concept of disagreement point does not exist in the other approaches, there is no gurantee that they do not violate it.



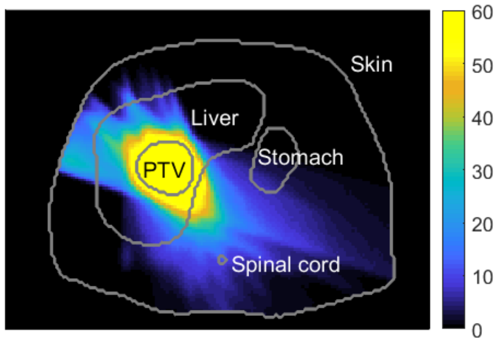
(a) Bargaining vs Lexico



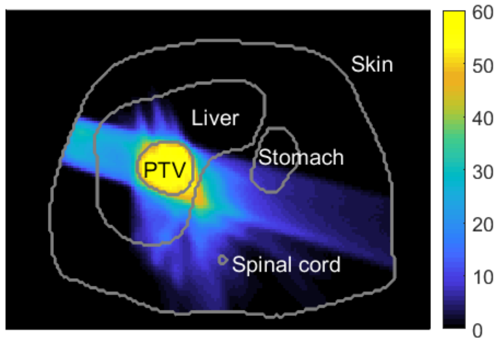
(b) Bargaining vs 2pϵc

Figure 5.7: Liver case: DVH comparison between the Nash optimal solution (solid curves) and Lexicographic optimization or 2pϵc approach (dashed curves)

By comparing Figures 5.7a and 5.7b, we observe that the plan generated by 2pϵc is significantly less extreme than the one generated by Lexico. While the plan generated by 2pϵc is favorable towards skin, we observe that its negative impacts on some organs (e.g., heart, spinal cord, and stomach) are more significant than the plan generated by the Nash bargaining solution. In other words, although the wish-list is designed to guide the Lexico and 2pϵc to create plans similar to the one generated by our proposed bargaining game, they ended up creating more extreme solutions compared to the proposed approach. This is mainly because as mentioned in Section 5.4, the proposed fluency map bargaining game naturally seeks to generate a Pareto-optimal plan that can balance the efficiency and fairness based on the parameters that users provide. However, such a process does not naturally exist in other solution approaches. So, users need to manually achieve that by exploring different parameters settings (e.g., different wish-lists). Finally, similar to the previous subsection, for interested readers, we report a 2D dose distribution map of the Nash optimal solution for both the Lexico and 2pϵc in Figure 5.8. We observe that similar conclusions to the above-mentioned ones can be made by comparing Figures 5.6d, 5.8a, and 5.8b.



(a) Lexico



(b) 2pϵc

Figure 5.8: Liver case: dose distribution map of the treatment plans generated by the Lexico and 2pϵc

Chapter 6: Solving Multiplicative Programs by Binary-encoding the Multiplication Operation

6.1 Introduction

The focus of this paper is on the following class of optimization problems, known as (linear) *multiplicative programs*,

$$\max \text{ or } \min \left\{ \prod_{i \in I} y_i(\boldsymbol{x}) : \; \boldsymbol{x} \in \mathcal{X}, \; \boldsymbol{y}(\boldsymbol{x}) \geqslant \boldsymbol{0} \right\}, \tag{6.1}$$

where $I := \{1, \ldots, p\}$, $\mathcal{X} \subseteq \mathbb{R}^{n_C} \times \mathbb{Z}^{n_I}$ ($n_C, n_I \geqslant 0$ being the number of continuous and integer variables, respectively) is the set of feasible solutions described by only linear constraints, and $\boldsymbol{y}(\boldsymbol{x})$ is a vector of p non-negative linear functions of $\boldsymbol{x} \in \mathcal{X}$. A multiplicative program is referred to as a *Maximum Multiplicative Program (MMP)* if it is in the form of maximization. Similarly, it is referred to as a *minimum Multiplicative Program (mMP)* if it is in the form of minimization. This study is motivated by two observations.

The first one is that MMPs and mMPs are typically studied independently in the existing body of literature as they are completely different in nature. This is highlighted by the fact that when there are no integer variables, i.e., $n_I = 0$, MMPs can be solved in polynomial time while mMPs remain NP-hard [3, 54, 55]. Consequently, not surprisingly, more effective solution methods exist for MMPs that cannot be used for solving mMPs [98]. One such method is to replace the multiplicative objective function by its corresponding log-transformed function, i.e., $\sum_{i \in I} \log(y_i(\boldsymbol{x}))$, and to solve the transformed problem by a mixed integer convex programming solver. Another method, which in practice is even faster, first replaces the multiplicative objective function by its corresponding geometric mean function, i.e., $\sqrt[p]{\prod_{i\in I} y_i(\boldsymbol{x})}$, and then transforms the problem into a mixed integer Second Order Cone Program (SOCP) in order to be solved by powerful commercial solvers such as CPLEX and Gurobi [3]. Overall, there is a lack of custom-built exact solution methods that can be used for both mMPs and MMPs, and this study attempts to address this gap.

The second observation is that for large values of p, the optimal objective value of Problem (6.1) tends to be either very large or very small due to the multiplicative nature of its objective function. We refer to this issue as *the curse of multiplication*. For example, suppose that $p = 100$ and $y_1 = \cdots = y_{100} = 99$ in an optimal solution. The optimal objective value, 99^{100}, is an astronomical number that far exceeds the capabilities of any state-of-the-art method to handle it. Moreover, for such magnitudes of objective values, the (typical) *relative* optimality gap tolerance that solvers use, e.g., 10^{-4}, is insufficient as the *absolute* optimality error continues to be beyond our imagination. Therefore, existing solvers need to select smaller values for their relative optimality gap tolerance, however, that can decrease their performance significantly and possibly create other forms of numerical issues. It is worth mentioning that transformation-based solution methods (e.g., the log-transformation) can possibly resolve the curse of multiplication for some multiplicative programs. However, transformations themselves can create other forms of numerical issues because of their non-linear nature and the fact that their outcomes can be irrational numbers. For example, $\log_{10} 99 = 2\log_{10} 3 + \log_{10} 11$ is obviously irrational. Overall, there is a lack of custom-built exact solution methods that can truly resolve the curse of multiplication, and this study attempts to address this gap.

6.1.1 Applications

An important application of multiplicative programs is in the field of conservation planning. This field deals with the issues related to preserving and/or increasing biodiversity [99]. Preserving biodiversity is crucial to human societies and the future of planet Earth, and its slow erosion constitutes a threat as consequential as that posed by climate change [100]. Preserving biodiversity is a vague goal and needs to be translated into measurable and definable objectives. Unfortunately, such a translation is not a trivial task and is known to be one of the significant challenges in environmental management [101, 33]. Therefore, a common approach in the literature of conservation planning is to compute extinction risks and use them for measuring biodiversity and comparing different solutions in conservation planning problems [33]. Specifically, the existing body of literature suggests that conservation planning problems need to be formulated and solved as multiplicative programs.

In such multiplicative programs, $\mathcal{X}$ represents the set of feasible solutions of a conservation planning problem. A feasible solution can represent, for example, which parcels of land can be purchased for protection, i.e., creating a nature reserve, based on the available budget as well as spatial and temporal constraints. Moreover, p represents the number of important/endangered species under consideration. Note that p can be a large number in practice as natural resource managers may be interested in considering dozens of species for a conservation planning problem [102]. For each species $i \in I$, let $f_i(\boldsymbol{x})$ be a (linear) function that captures the extinction risk corresponding to solution $\boldsymbol{x} \in \mathcal{X}$. Based on these notations, the literature suggests that mMPs can be solved if managers are interested in minimizing the probability of more species being extinct. For mMPs, we should set $y_i(\boldsymbol{x}) = f_i(\boldsymbol{x})$ for each species $i \in I$. The literature also suggests that MMPs can be solved if managers are interested in minimizing the probability of all species being extinct. Then, for MMPs, we should set $y_i(\boldsymbol{x}) = 1 - f_i(\boldsymbol{x})$ for each species $i \in I$.

In addition to conservation planning, multiplicative programs have applications in many other domains. For example, mMPs have applications in bond portfolio management, economic analysis, and VLSI chip design [103]. MMPs, on the other hand, can be used to compute the Nash solution in bargaining problems in the field of cooperative game theory [104]. In bargaining problems, the multiplicative function is sometimes referred to as the *Nash Social Welfare function* [105]. Hence, in bargaining problems, MMPs can be viewed as maximizing the Nash Social Welfare function over the feasible set of actions. Note that many real-world bargaining problems require integer decision variables, e.g., the allocation of indivisible goods among heirs [105]. MMPs can also be used in computing an equilibrium for Fisher's linear or Kelly's capacity allocation market [106] or systems reliability problems [107, 108, 109].

Finally, we note that multiplicative programs have applications in multi-objective optimization [3, 55]. Two areas of research (among others) in multi-objective optimization are (1) computing the entire Pareto-optimal/efficient frontier [110, 111] and then asking decision makers to select a desirable solution; (2) optimization over the Pareto-optimal/ efficient set whose attempts to directly compute a desirable solution without generating the entire Pareto-optimal frontier [112]. Multiplicative programs are special cases of the latter. Observe that for MMPs (mMPs), $y_i(\boldsymbol{x})$ can be viewed as an individual objective function that needs to be maximized (minimized) for each $i \in I$. Therefore, the multi-objective optimization counterpart of a multiplicative program can be stated as

$$\max \ \text{ or } \ \min \{y_1(\boldsymbol{x}), \ldots, y_p(\boldsymbol{x}) : \ \boldsymbol{x} \in \mathcal{X}, \ \boldsymbol{y}(\boldsymbol{x}) \geqslant \boldsymbol{0}\}.$$

In this case, solving MMPs (mMPs) is equivalent to maximizing (minimizing) the super-criterion $\prod_{i \in I} y_i(\boldsymbol{x})$ over the set of Pareto-optimal solutions of its multi-objective optimization counterpart [98, 113, 114, 115, 116].

6.1.2 Contributions

The main contribution of our research is to develop techniques in order to binary-encode the multiplication operation in the objective function of a multiplicative program. We note that the idea of binary-encoding has a long history in the field of mathematical optimization and computer science, see for instance [117, 118, 119, 120] and [121]. In fact, it is the very foundation of any existing (non-quantum) computer. We are therefore certainly not the first to apply the binary-encoding operation – however, to the best of our knowledge, we are the first to employ it for developing techniques for solving multiplicative programs.

Using our proposed techniques, a family of novel solution methods will be introduced for solving both MMPs and mMPs. The new family of solution methods can resolve the curse of multiplication as they do not necessarily need to deal with the magnitude of the objective values. In fact, they are designed to compute the value of bits of the optimal objective value from the most significant bit to the least significant one. Overall, a nice property of the proposed family of solution methods is that they only solve a finite number of integer linear programs to compute an optimal solution for a multiplicative program. Depending on a specific solution method, the number of integer linear programs that need to be solved is at least one and at most equal to the number of bits required to represent the objective of a multiplicative program. Solution methods in our proposed family are different mainly because of two reasons.

The first reason is that the binary-encoding of the multiplication operation can be done in multiple ways (if $p > 2$) which in turn results in different equivalent reformulations. In this study, we explore two categories of reformulations which we will refer to as *Altogether* and *Nested*. The Altogether strategy generates a reformulation that directly binary-encodes $\prod_{i \in I} y_i$ as a whole. To run the Nested strategy, first, a sequence will be generated, e.g., $(1, 2, 3, \dots, p)$, reflecting how the multiplication process should be carried out. Then, in the reformulation, $p - 1$ binary-encoded multiplications of two variables will be generated based on the sequence. For example, for the sequence $(1, 2, 3, \dots, p)$, the reformulation contains the binary-encoded multiplication $y_1 \times y_2$, the binary-encoded multiplication $y' \times y_3$ (where y' is the outcome of $y_1 \times y_2$), and so on.

The second reason is that after creating a reformulation, different search mechanisms can be developed to solve it. In this study, we consider two main categories of search mechanisms which we will refer to as *One-shot* and *Bitwise*. One-shot means that the reformulation should be solved directly by solving a mixed integer linear program. In this strategy, the optimal values of all bits will be obtained at the same time. For the Bitwise reformulation, the values of bits will be obtained iteratively, i.e., exploring one bit at a time. To compute the optimal value of each bit, one mixed integer linear program must be solved (if needed). We also develop two cut-generating techniques that can be combined with the Bitwise strategy to tighten the reformulations in each iteration and possibly reduce the number of mixed integer linear programs needed.

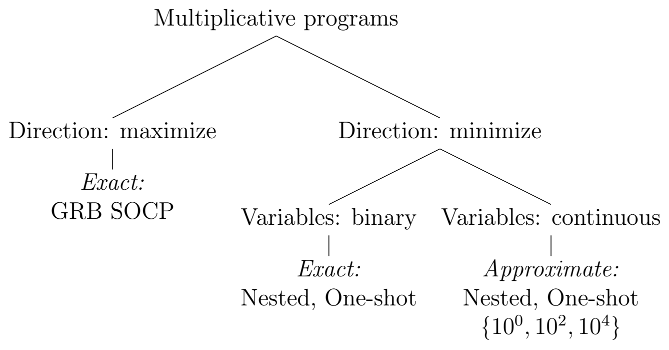


Figure 6.1: High-level summary of computational results as a tree (recommended solution approaches)

The Bitwise strategy is naturally slower as it is the only strategy that fully resolves the curse of multiplication. The One-shot strategy, on the other hand, is faster as it does not resolve the curse of multiplication but directly solves an integer linear programming reformulation of a multiplicative program. In this study, we first provide some numerical examples to show that the Bitwise strategy is the best strategy to deal with large values of p as it does not run into any numerical instabilities. We then conduct a detailed computational study on instances with $p \in \{2, 3, 4\}$ and compare the One-shot strategy with GRB SOCP (Gurobi mixed integer SOCP solver) on MMPs and GRB Nonconvex (Gurobi mixed integer nonconvex quadratic solver) on mMPs. The computational study provides a practical guide indicating which solution method to use for which problem variation. A high-level summary of this guide is shown in Figure 6.1, which is a tree representation of the results later shown in Section 6.5, Table 6.5. In this tree, for a given problem variation, we list the solution methods with the shortest solution times. Since the aforementioned applications of multiplicative programs are typically either pure binary or pure continuous cases, we employ two classes of (random) instances in our study. We note that to binary-encode the multiplication operation, $y_i(\boldsymbol{x})$ should only take integer values for all $i \in I$. Otherwise, our proposed family of solution methods are only approximate methods. For such instances, the function $y_i(\boldsymbol{x})$ will be assumed to be integer for all $i \in I$ after being multiplied by a sufficiently large number. The sufficiently large number should be determined based on the level of precision required. In our computational study, we consider three possible scenarios for the value of the multiplier: 10^0, 10^2, and 10^4. Overall, our results show that for MMPs, GRB SOCP is the best choice. For mMPs, however, our proposed family of solution methods beat the off-the-shelf benchmark most of the time.

6.1.3 Outline

The remainder of this chapter has the following outline: in Section 6.2, we introduce the methodology at a high level, and in Section 6.3, we introduce the methodology in detail. In Section 6.4, we show the power of the Bitwise search mechanism in resolving the curse of multiplication for some extreme examples. In Section 6.5, we compare the solution methods with the aforementioned benchmarks in an extensive computational study that we ran on 900 instances spanning a total of 5,400 runs grouped in 4 experiments, with each experiment designed to find the best solution method for a given problem variation. Finally, in Section 7.5, we provide the concluding remarks of the paper as well as some final recommendations.

6.2 High-level Methodology

A multiplicative program can be stated as

$$\max_{\boldsymbol{x},\boldsymbol{y}} \text{ or } \min \quad \prod_{i\in I} y_i \tag{6.2}$$

$$\text{s.t.} \quad \boldsymbol{y} = D\boldsymbol{x} + \boldsymbol{d}, \tag{6.3}$$

$$A\boldsymbol{x} \leqslant \boldsymbol{b}, \tag{6.4}$$

$$\boldsymbol{x}, \boldsymbol{y} \geqslant \boldsymbol{0}, \ \boldsymbol{x} \in \mathbb{R}^{n_C \times n_I}, \ \boldsymbol{y} \in \mathbb{R}^{p_C \times p_I}, \tag{6.5}$$

where n_C and n_I denote the number of continuous and integer decision variables in the space of x-variables, respectively. Moreover, p_C and p_I denote the number of continuous and integer decision variables in the space of y-variables, respectively. Given $n := n_C + n_I$ and $p := p_C + p_I$, D is a $p \times n$ matrix representing the coefficients of the multiplicative terms. Similarly, $\boldsymbol{d}$ is a vector of size p representing the constants of the multiplicative terms. Further, A is an $m \times n$ matrix that denotes the technological coefficients, with m being the number of linear constraints. Finally, $\boldsymbol{b}$ represents the m-sized vector of right-hand side values. A powerful off-the-shelf solver that can solve any multiplicative program is Gurobi. For an MMP, users need to transform the problem into a (mixed integer) SOCP as described in [51, 3] and then call GRB SOCP (Gurobi mixed integer SOCP solver) to solve it. To solve mMPs, users can use GRB Nonconvex (Gurobi mixed integer nonconvex quadratic solver). Obviously, for $p > 2$, an mMP is not quadratic. However, it can be brought into quadratic form by introducing some auxiliary variables. For example, instead of $y_1y_2y_3$, one can write $y'y_3$ where $y' = y_1y_2$.

The breakthrough idea of the proposed research is to binary-encode the multiplication operation (analogously to how a computer conducts it internally) such that the objective value of any solution is represented by bits. In that case rather than dealing with the entire optimal objective value, computing the bits of the optimal objective value becomes crucial.

Each bit is a binary decision variable and the values of the bits can be determined in an iterative fashion from the most significant bit to the least significant one. In each iteration, the value of one bit of the optimal objective value will be computed and then its value will be fixed for future iterations. Describing the full mathematical model of a binary-encoded reformulation of the multiplicative program requires heavy notation. Instead of introducing the full notation now (we will do so later), in this section, we present the key idea using a simple example with $p_I = 2$ and $p_C = 0$. Suppose that it is known that $y_1, y_2 \leqslant 10$. To develop a binary-encoded reformulation, the following steps need to be taken:

- *Step 1:* Find an upper bound for the optimal objective value. Observe that, by defining $z = y_1 y_2$, we have $z \leqslant 100$. This implies that the binary representation of the value of the product has at most $\lfloor \log_2 100 \rfloor + 1 = 7$ bits. Let $\hat{z}_0, \dots, \hat{z}_6$ denote the values of the bits to binary-encode z where $\hat{z}_0$ is the least significant bit and $\hat{z}_6$ is the most significant one. The optimal values of $\hat{z}_0, \dots, \hat{z}_6$ are unknown and the idea is to compute them one by one. As an example, an illustration of how to compute $\hat{z}_5$ is shown in Figure 6.2 where we binary-encode the multiplication of 14 and 5.

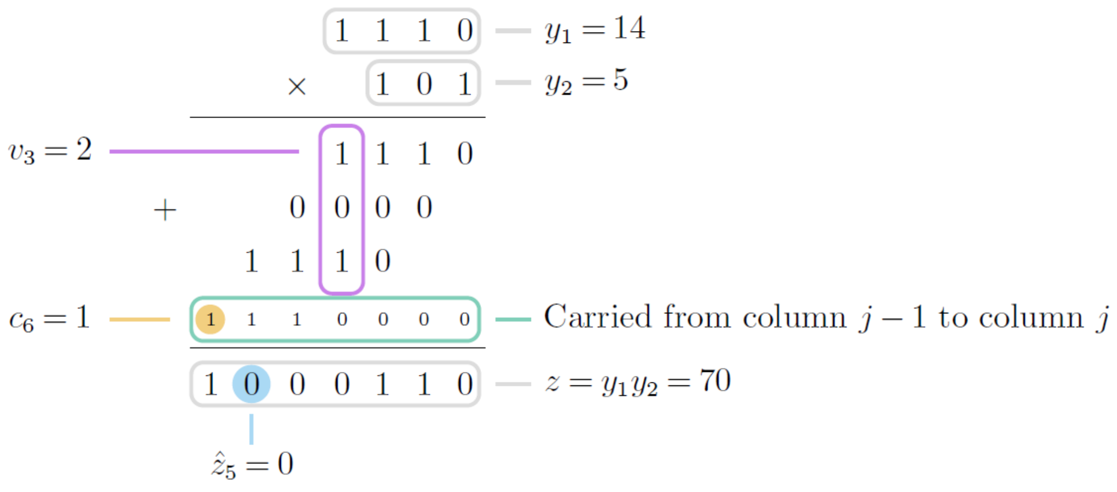


Figure 6.2: Binary multiplication of two binary-encoded numbers (numbering/indexing starts from zero and is done from right to left, e.g., column $j-1$ is to the right of column j and $\hat{z}_4$ is to the right of $\hat{z}_5$)

- *Step 2:* Represent y_1 and y_2 in a binary-encoded format. This can be done easily (using linear equations) when it is known that both y_1 and y_2 can only take integer values:

$$y_1 = \sum_{i=0}^{\lfloor \log_2 10 \rfloor + 1} 2^i \hat{y}_{1,i} \quad \text{and} \quad y_2 = \sum_{i=0}^{\lfloor \log_2 10 \rfloor + 1} 2^i \hat{y}_{2,i} \tag{6.6}$$

  where $\hat{y}_{1,i}$ and $\hat{y}_{2,i}$ capture the value of the bit with index number i for y_1 and y_2, respectively. Note that $\lfloor \log_2 10 \rfloor + 1 = 4$, and hence, only 4 bits each are required to represent y_1 and y_2.

- *Step 3:* Apply the principles of the *binary multiplication* (see Figure 6.2) for capturing the value of $\hat{z}_j$ for all $j \in \{0, 1, \ldots, 6\}$. As an aside, we note that the 'binary multiplication' is basically the *Fast Fourier Transform (FFT) based multiplication of integers* [122], which is closely related to a multiplication method commonly referred to as *ancient Egyptian multiplication* or *Ethiopian/Russian multiplication*[6.1].

  - The sum of all binary products of column j is denoted by v_j and it can be captured by the following equation for all $j \in \{0, \ldots, 6\}$:

$$v_j = \sum_{\substack{i,i' \in \{0,1,2,3\}: \\ i+i'=j}} \hat{y}_{1,i} \ \hat{y}_{2,i'}. \tag{6.7}$$

  - The carried value from column $j-1$ to j is denoted by c_j and it can be captured by the following equation for all $j \in \{0, \ldots, 6\}$:

$$c_j = \begin{cases} \left\lfloor \dfrac{v_{j-1} + c_{j-1}}{2} \right\rfloor & \text{if } j \neq 0 \\ 0 & \text{if } j = 0 \end{cases}. \tag{6.8}$$

[6.1] As the name suggests, such multiplication methods were already known to the Ancient Egyptians, as can be seen e.g. in *The Rhind Mathematical Papyrus* dating to around 1550 BC (see the *British Museum* objects EA10057 and EA10058).

– Finally, the value of $\hat{z}_j$ can be captured by the following equation for all $j \in \{0, \dots, 6\}$:

$$\hat{z}_j = \begin{cases} v_j + c_j - 2c_{j+1} & \text{if } j \neq 6 \\ v_j + c_j & \text{if } j = 6 \end{cases}. \tag{6.9}$$

An illustration of how to compute v_3, c_6, and $\hat{z}_5$ is shown in Figure 6.2. In light of the above, the binary-encoded reformulation of the multiplicative programs for this example can be stated as

$$\begin{aligned} \max_{\boldsymbol{x},\boldsymbol{y},\hat{\boldsymbol{z}},\hat{\boldsymbol{y}},\boldsymbol{c},\boldsymbol{v}} \text{ or } \min \quad & \sum_{j=0}^{6} 2^j \hat{z}_j \\ \text{s.t.} \quad & \text{(6.3)–(6.5) and (6.6)–(6.9)}, \\ & \hat{y}_{1,i}, \hat{y}_{2,i} \in \{0,1\}, \qquad\qquad i \in \{0,1,\dots,3\}, \\ & \hat{z}_j \in \{0,1\}, \text{ and } c_j, v_j \in \mathbb{N}_0 \quad j \in \{0,1,\dots,6\}, \end{aligned} \tag{6.10}$$

where $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. Note that, in Problem (6.10), the variable $\hat{z}_j$ represents the bit $j \in \{0, 1, \dots, 6\}$ of the objective value and it naturally takes a binary value. Observe that in Problem (6.10), only Constraint (6.7) is nonlinear. However, this constraint can be easily linearized. In Constraint (6.7), the term $\hat{y}_{1,i}\hat{y}_{2,i'}$ is nonlinear but $\hat{y}_{1,i}$ and $\hat{y}_{2,i'}$ are binaries. Hence, the term can be replaced by a new binary variable $u_{i,i'}$ and by adding the following three inequalities (often referred to as *McCormick envelopes* [123, 124]):

$$u_{i,i'} \leqslant \hat{y}_{1,i}, \qquad u_{i,i'} \leqslant \hat{y}_{2,i'}, \qquad u_{i,i'} \geqslant \hat{y}_{1,i} + \hat{y}_{2,i'} - 1.$$

Note that it is not necessary to impose the integrality condition on $u_{i,i'}$ as it naturally takes integer values. Also, note that Constraint (6.8) appears nonlinear but can be written in linear form without introducing any new variables as follows:

$$\begin{aligned}
&c_0 = 0, && \\
&2c_j \geqslant v_{j-1} + c_{j-1} - 1, && j \in \{1, \ldots, 6\}, \\
&2c_j \leqslant v_{j-1} + c_{j-1}, && j \in \{1, \ldots, 6\}.
\end{aligned}$$

Overall, the proposed reformulation is a mixed integer linear program that can be solved directly by commercial solvers such as CPLEX and Gurobi. By directly solving the proposed reformulation, the optimal values of all bits will be determined at the same time. However, solving the proposed binary reformulation directly does not resolve the curse of multiplication as the magnitude of the objective value of the reformulation remains exactly the same as in the original formulation. Observe now that it is not necessary to solve the binary-encoded reformulation directly. Instead, the values of the bits of the optimal objective value can be obtained in an iterative fashion from the most significant bit to the least significant one. More formally, at iteration $t = 0, \ldots, 6$, Problem (6.11) needs to be solved:

$$\begin{aligned}
\max_{\boldsymbol{x},\boldsymbol{y},\hat{\boldsymbol{z}},\hat{\boldsymbol{y}},\boldsymbol{c},\boldsymbol{v}} \text{ or } \min \quad & \hat{z}_{6-t} && \\
\text{s.t.} \quad & \hat{z}_{6-t'} = \hat{z}^*_{6-t'}, && t' \in \{0, \ldots, t-1\}, \\
& \text{(6.3)–(6.5) and (6.6)–(6.9)}, && \\
& \hat{y}_{1,i}, \hat{y}_{2,i} \in \{0, 1\}, && i \in \{0, 1, \ldots, 3\}, \\
& \hat{z}_j \in \{0, 1\}, \text{ and } c_j, v_j \in \mathbb{N}_0, && j \in \{0, 1, \ldots, 6\},
\end{aligned} \tag{6.11}$$

Note that in Problem (6.11), $\hat{z}^*_{6-t'}$ represents the optimal value of Problem (6.11) obtained in iteration t'. In other words, after each iteration, the value of its corresponding bit will be fixed for future iterations. Observe that the proposed iterative method completely resolves the curse of multiplication. Moreover, users can impose any desired level of precision as the termination condition on the proposed algorithm. The level of precision then defines how many bits need to be accurately computed. Overall, in the proposed iterative method only a mixed integer linear program needs to be solved in each iteration whose objective value is either zero or one (assuming the original formulation is feasible). In Section 6.3.2.2, we discuss that it is not necessary to solve *all* mixed integer linear programs as the optimal value of some bits can be determined directly based on the information obtained in the previous iterations.

In summary, from the discussions above, after developing a binary-encoded reformulation for a multiplicative program, two search mechanisms can be used to solve it: the direct approach, One-shot, and the iterative one, Bitwise. We note that in the simple example above, we assumed that $p_C = 0$. However, this is not an impractical assumption since if $p_C > 0$, it can still be approximated to any desired level of precision by a multiplicative program with $p_C = 0$. Specifically, we can assume that only a certain number of digits after the decimal point is needed/necessary for fractional y-variables. If, for example, two digits after the decimal point are needed for y_i, then we can still assume that y_i is an integer variable after multiplying the right-hand-side of its associated constraint in (6.3) by 100. Such transformations are of course costly as they will increase the maximum number of bits required to represent the optimal objective value of a multiplicative program. However, they can be used to solve multiplicative programs involving fractional y-variables. We note that in terms of implementation, there are a few points that should be considered when dealing with continuous variables. We will address these in Section 6.3.4.

Finally, we note that while for instances with $p = 2$, there is only one way to develop a binary-encoded reformulation, for $p > 2$ there are many possible ways. In this study, we focus only on two main categories of reformulations: Nested and Altogether. For example, if $p = 3$, a Nested reformulation is $(y_1y_2)y_3$ and the Altogether reformulation is $(y_1y_2y_3)$. Such binary-encoded reformulations are substantially different in terms of the number of variables and constraints that they introduce. For example, for $p = 3$, the Altogether reformulation comes with the disadvantage of making the linearization of Equation (6.7) more difficult because instead of bilinear terms, trilinear terms will appear in Equation (6.7). However, it also comes with the advantage of having only minor impacts on increasing the complexity of Equations (6.6), (6.8), and (6.9). The Nested reformulation, on the other hand, has exactly the opposite effect. Later, in Section 6.3.1.2, we show that a nice property of the Nested reformulation is that it keeps the size of the problem polynomially bounded while the Altogether reformulation does not guarantee that.

Table 6.1: Frequently used notations (subscripts and superscripts will be added whenever needed)

Variables	
z	An integer variable used for representing the product of some y variables
$\hat{z}$	A binary variable used for representing a bit of the binary-encoded version of a z variable
$\hat{y}$	A binary variable used for representing a bit of the binary-encoded version of a y variable
v	An integer variable used for representing the summation of a column in the process of binary multiplication
c	An integer variable used for representing the carried value from a column to its next column in the process of binary multiplication
u	A binary variable used for representing McCormick envelopes of the product of some $\hat{y}$ variables
Parameters	
n^z	An integer parameter showing the maximum number of bits required to represent a z variable
n^y	An integer parameter showing the maximum number of bits required to represent a y variable

6.3 Detailed Methodology

In this section, we provide the detailed description of our proposed family of solution methods for multiplicative programs. Our notational convention in this section is summarized in Table 6.1. We start this section by assuming $p_C = 0$ for now, but we will later relax this condition.

6.3.1 Binary-encoded Reformulations

6.3.1.1 Products of Length Two: The Nested *Reformulation*

We are now going to implement the FFT-based multiplication as constraints to reformulate a multiplicative program. For this and for the following section, let

- $\overline{y}_i$ denote the upper bounds on y_i, $i \in I$ (calculated in preprocessing),
- $n_i^y = \lfloor \log_2(\overline{y}_i) \rfloor + 1$ denote the number of binary variables needed to encode y_i, $i \in I$,

- $J_i^y = \{0, \dots, n_i^y - 1\}$, $i \in I$,

- $\hat{y}_{i,j} \in \{0, 1\}$, $j \in J_i^y$, denote the binary variables to encode y_i, $i \in I$.

In this formulation, the idea is to interleave (or *nest*) the product (6.2) into products of length two – that is – for all $i \in I$, we set

$$z_i = y_i \times \begin{cases} 1, & i = 1, \\ z_{i-1}, & i \in I \backslash \{1\}, \end{cases}$$

where (6.2) $= \prod\limits_{i \in I} y_i = z_p$. Let

- $\overline{z}_i = \prod\limits_{\substack{i' \in I: \\ i' \leqslant i}} \overline{y}_{i'}$ denote the upper bounds on z_i, $i \in I$,

- $n_i^z = \lfloor \log_2(\overline{z}_i) \rfloor + 1$ denote the number of binary variables needed to encode z_i, $i \in I$,

- $J_i^z = \{0, \dots, n_i^z - 1\}$, $i \in I$,

- $\hat{z}_{i,j} \in \{0, 1\}$, $j \in J_i^z$, denote the binary variables to encode z_i, $i \in I$,

- $u_{i,j,j'} \in [0, 1]$, $i \in I \backslash \{1\}$, $j \in J_i^y$, $j' \in J_{i-1}^z$, be a continuous variable that is naturally binary and that encodes the $\hat{y}\hat{z}$ products,

- $v_{i,j} \in \mathbb{N}_0$, $i \in I \backslash \{1\}$, $j \in J_i^z$, be an integer variable that represents the j-th column sum of the u variables,

- $c_{i,j} \in \mathbb{N}_0$, $i \in I \backslash \{1\}$, $j \in J_i^z$, be an integer variable that represents the carried value from column $j - 1$ to column j.

We reformulate a multiplicative program *linearly exactly* as follows, referring to it as the Nested reformulation:

$$\max_{\boldsymbol{x},\boldsymbol{y},\hat{\boldsymbol{z}},\hat{\boldsymbol{y}},\boldsymbol{c},\boldsymbol{v},\boldsymbol{u}} \text{ or } \min \sum_{j \in J_p^z} 2^j \hat{z}_{p,j}$$

$$\begin{aligned}
\text{s.t. } & (6.3), (6.4), (6.5), && \\
& y_i = \sum_{j \in J_i^y} 2^j \hat{y}_{i,j}, && i \in I \setminus \{1\}, && (6.12) \\
& y_1 = \sum_{j \in J_1^z} 2^j \hat{z}_{1,j}, && && (6.13) \\
& u_{i,j,j'} \geqslant \hat{y}_{i,j} + \hat{z}_{i-1,j'} - 1, && i \in I \setminus \{1\},\ j \in J_i^y,\ j' \in J_{i-1}^z, && (6.14) \\
& u_{i,j,j'} \leqslant \hat{y}_{i,j}, && i \in I \setminus \{1\},\ j \in J_i^y,\ j' \in J_{i-1}^z, && (6.15) \\
& u_{i,j,j'} \leqslant \hat{z}_{i-1,j'}, && i \in I \setminus \{1\},\ j \in J_i^y,\ j' \in J_{i-1}^z, && (6.16) \\
& v_{i,j} = \sum_{\substack{j' \in J_i^y,\, j'' \in J_{i-1}^z: \\ j' + j'' = j}} u_{i,j',j''}, && i \in I \setminus \{1\},\ j \in J_i^z, && (6.17) \\
& c_{i,0} = 0, && i \in I \setminus \{1\}, && (6.18) \\
& 2c_{i,j} \geqslant v_{i,j-1} + c_{i,j-1} - 1, && i \in I \setminus \{1\},\ j \in J_i^z \setminus \{0\}, && (6.19) \\
& 2c_{i,j} \leqslant v_{i,j-1} + c_{i,j-1}, && i \in I \setminus \{1\},\ j \in J_i^z \setminus \{0\}, && (6.20) \\
& \hat{z}_{i,j} = v_{i,j} + c_{i,j} - \begin{cases} 2c_{i,j+1}, & j < n_i^z - 1, \\ 0, & j = n_i^z - 1, \end{cases} && i \in I \setminus \{1\},\ j \in J_i^z, && (6.21) \\
& u_{i,j,j'} \in [0,1], && i \in I \setminus \{1\},\ j \in J_i^y,\ j' \in J_{i-1}^z, && (6.22) \\
& v_{i,j}, c_{i,j} \in \mathbb{N}_0, && i \in I \setminus \{1\},\ j \in J_i^z, && (6.23) \\
& \hat{y}_{i,j} \in \{0,1\}, && i \in I \setminus \{1\},\ j \in J_i^y, && (6.24) \\
& \hat{z}_{i,j} \in \{0,1\}, && i \in I,\ j \in J_i^z. && (6.25)
\end{aligned}$$

Note that Constraints (6.14)–(6.17) are equivalent to computing the *cyclic convolution* (i.e., Equation (6.7)) using a McCormick relaxation. The u variables will naturally take binary values. Constraints (6.18)–(6.20) are added to capture the carries (i.e., they are the linear form of Equation (6.8)), and Constraints (6.21) compute the binary-encoded product. Observe that the way we interleave the product (6.2) into

$$\underbrace{(\underbrace{(\cdots\underbrace{(\underbrace{(y_1)}_{=z_1}\, y_2)}_{=z_2}\cdots y_{p-1})}_{=z_{p-1}}\, y_p)}_{=z_p}$$

raises an interesting side question: what is the *optimal* way of reindexing the y variables? We leave this question as a future research direction and we do not apply reindexing in this study. However, the following proposition shows that reindexing may result in Nested reformulations with different numbers of variables (and constraints) if $n_1^y = \ldots = n_p^y$ does not hold. The proof of this and all the following propositions in this study can be found in Appendix E1.

Proposition 6.1. *For a given reindex $\sigma : (1, \ldots, p) \to (1, \ldots, p)$, the number of u variables, i.e., continuous variables, in the Nested reformulation is*

$$\sum_{i=2}^{p} \left(n_{\sigma(i)}^{y} \left(\left\lfloor \sum_{j=1}^{i-1} \log_2 \left(\bar{y}_{\sigma(j)} \right) \right\rfloor + 1 \right) \right),$$

and the total number of v and c variables, i.e., integer variables, in the Nested reformulation is

$$2 \left(\sum_{i=2}^{p} \left(\left\lfloor \sum_{j=1}^{i} \log_2 \left(\bar{y}_{\sigma(j)} \right) \right\rfloor + 1 \right) \right).$$

6.3.1.2 Products of Length p: the Altogether *Reformulation*

Interleaving the product (6.2) into products of length two, while convenient, is not necessary. We may also reformulate the product *altogether*. A motivating example is shown in Table 6.2 where we carry out the *generalized* FFT-based multiplication of the three integers $y_1 = y_2 = y_3 = 7$.

Table 6.2: Generalized FFT-based multiplication of the three integers $y_1 = y_2 = y_3 = 7$

j	8	7	6	5	4	3	2	1	0
$\hat{y}_{1,j}$	0	0	0	0	0	0	1	1	1
$\hat{y}_{2,j}$	0	0	0	0	0	0	1	1	1
$\hat{y}_{3,j}$	0	0	0	0	0	0	1	1	1
							1	1	1
						1	1	1	
					1	1	1		
						1	1	1	
					1	1	1		
				1	1	1			
					1	1	1		
				1	1	1			
			1	1	1				
v_j	0	0	1	3	6	7	6	3	1
c_j	1	2	4	5	5	3	1	0	0
$\hat{z}_j$	1	0	1	0	1	0	1	1	1
$y_1y_2y_3 = 2^8 + 2^6 + 2^4 + 2^2 + 2^1 + 2^0$									

We are now going to implement the *generalized* FFT-based multiplication as constraints to reformulate a multiplicative program. Let

- $\overline{z} = \prod\limits_{i \in I} \overline{y}_i$ denote the upper bound on $z = \prod\limits_{i \in I} y_i$,
- $n^z = \lfloor \log_2(\overline{z}) \rfloor + 1$ denote the number of binary variables needed to encode z,
- $J^z = \{0, \ldots, n^z - 1\}$,
- $\mathcal{P}_k$, $k \in J^z$, denote the set of two-dimensional sets whose j-indices add up to k:

$$\mathcal{P}_k = \left\{ \{(1, j_1),\ (2, j_2),\ \ldots,\ (p, j_p)\} :\ j_i \in J_i^y,\ i \in I;\ \sum_{i \in I} j_i = k \right\},$$

As an example, consider Table 6.2 and let $p = 3$ and $\bar{y}_i = y_i = 7$ for $i \in I$. We then have $n_i^y = 3$ and $J_i^y = \{0, 1, 2\}$ for $i \in I$. For $\bar{z} = \prod_{i \in I} \bar{y}_i = 343$, we then have $n^z = 9$

and $J^z = \{0, \ldots, 8\}$. Therefore,

$$\begin{aligned}
\mathcal{P}_0 &= \{\{(1,0),(2,0),(3,0)\}\},\\
\mathcal{P}_1 &= \{\{(1,1),(2,0),(3,0)\},\ \{(1,0),(2,1),(3,0)\},\ \{(1,0),(2,0),(3,1)\}\},\\
\mathcal{P}_2 &= \{\{(1,2),(2,0),(3,0)\},\ \{(1,0),(2,2),(3,0)\},\ \{(1,0),(2,0),(3,2)\},\\
&\qquad \{(1,1),(2,1),(3,0)\},\ \{(1,1),(2,0),(3,1)\},\ \{(1,0),(2,1),(3,1)\}\},\\
&\quad \vdots\\
\mathcal{P}_8 &= \varnothing.
\end{aligned}$$

Note that, intuitively speaking, each $k \in J^z$ basically represents the index of a column of values (which are all 1's in this example) in the large middle section of Table 6.2. Specifically, index $k = 0$ is the index of the column on the rightmost side and $k = 8$ is the index of the column on the leftmost side. Since the number of values reported in the middle section for index $k = 0$ is one, $\mathcal{P}_0$ will only have one element. Similarly, since the number of values reported in the middle section for index $k = 1$ is three, $\mathcal{P}_1$ will only have three elements.

- $\mathcal{P} = \bigcup_{k \in J^z} \mathcal{P}_k$,
- $\hat{z}_j \in \{0,1\}$, $j \in J^z$, denote the binary variables to encode z,
- $u_P \in [0,1]$, $P \in \mathcal{P}$, be a continuous variable that is naturally binary and that encodes the $\hat{y}_{i,j}$ products, $(i,j) \in P$,
- $v_j \in \mathbb{N}_0$, $j \in J^z$, be an integer variable that represents the j-th column sum of the u variables,
- $c_j \in \mathbb{N}_0$, $j \in J^z$, be an integer variable that represents the carried value from column $j-1$ to column j.

We now reformulate a multiplicative program *linearly exactly* as follows, referring to it as the Altogether reformulation:

$$\max_{\boldsymbol{x},\boldsymbol{y},\hat{\boldsymbol{z}},\hat{\boldsymbol{y}},\boldsymbol{c},\boldsymbol{v},\boldsymbol{u}} \text{ or } \min \sum_{j \in J^z} 2^j \hat{z}_j$$

$$\begin{aligned}
\text{s.t. } & (6.3),\ (6.4),\ (6.5), && \\
& y_i = \sum_{j \in J_i^y} 2^j \hat{y}_{i,j}, && i \in I, && (6.26)\\
& u_P \geqslant \sum_{(i,j) \in P} \hat{y}_{i,j} - (p-1), && P \in \mathcal{P}, && (6.27)\\
& u_P \leqslant \hat{y}_{i,j}, && P \in \mathcal{P},\ (i,j) \in P, && (6.28)\\
& v_j = \sum_{P \in \mathcal{P}_j} u_P, && j \in J^z, && (6.29)\\
& c_0 = 0, && && (6.30)\\
& 2c_j \geqslant v_{j-1} + c_{j-1} - 1, && j \in J^z \backslash \{0\}, && (6.31)\\
& 2c_j \leqslant v_{j-1} + c_{j-1}, && j \in J^z \backslash \{0\}, && (6.32)\\
& \hat{z}_j = v_j + c_j - \begin{cases} 2c_{j+1}, & j < n^z - 1, \\ 0, & j = n^z - 1, \end{cases} && j \in J^z, && (6.33)\\
& u_P \in [0,1], && P \in \mathcal{P}, && (6.34)\\
& \hat{z}_j \in \{0,1\}, \text{ and } v_j, c_j \in \mathbb{N}_0, && j \in J^z, && (6.35)\\
& \hat{y}_{i,j} \in \{0,1\}, && i \in I,\ j \in J_i^y. && (6.36)
\end{aligned}$$

Proposition 6.2. *The number of u variables, i.e., continuous variables, in the Altogether reformulation is*

$$\prod_{i \in I} n_i^y,$$

and the total number of v and c variables, i.e., integer variables, in the Altogether reformulation is

$$2\left(\left\lfloor \sum_{i\in I}\log_2\left(\bar{y}_i\right)\right\rfloor + 1\right) = 2n^z.$$

Comparing Propositions 6.1 and 6.2 shows that the Nested and Altogether reformulations have very different properties in terms of the number of variables (and constraints). Let $L := \max\{n_1^y, \dots, n_p^y\}$ and assume that the size of a multiplicative program is polynomially bounded. In that case, the size of its Nested reformulation is polynomially bounded, too, as the number of its u variables is $O(L^2p^2)$ and the total number of its c and v variables is $O(2Lp^2)$. However, the size of the Altogether reformulation is exponential as the number of its u variables is $O(L^p)$ and the total number of its c and v variables is $O(2Lp)$.

6.3.2 Search Mechanisms

To optimize the Nested and Altogether reformulations, we propose two search mechanisms, which in this section we refer to as *One-shot* and *Bitwise.*

6.3.2.1 One-shot Search Mechanism

In this search mechanism, we directly solve the Nested or Altogether reformulation, which are (mixed) integer linear programs, using an off-the-shelf solver such as CPLEX or Gurobi. The advantage of this search mechanism is that it only solves one optimization problem to compute the optimal values of all bits of the multiplicative objective function. Its disadvantage is that it does not resolve the curse of multiplication.

6.3.2.2 Bitwise Search Mechanism

Without loss of generality, in this section, we explain the Bitwise search mechanism only in the context of the Altogether reformulation. To apply it in the context of the Nested reformulation, we should add the subscript p to some notations used in this section.

Specifically, any instance of n^z, J^z, $\hat{z}_i$, $\hat{z}_i^*$, and $\hat{z}_i^t$ should be changed to n_p^z, J_p^z, $\hat{z}_{p,i}$, $\hat{z}_{p,i}^*$, and $\hat{z}_{p,i}^t$, respectively.

In the Bitwise search mechanism, we compute the optimal values of bits one by one from the most significant one to the least significant one. In each iteration, the optimal value of one bit will be computed and will be fixed for future iterations. The advantage of this search mechanism is that it fully resolves the curse of multiplication. Its disadvantage is that one (mixed) integer integer program with the optimal objective value of either zero or one must be solved for each bit. Specifically, for the Altogether reformulation, the following objective function must be optimized at iteration $t \in J^z$,

$$\max \text{ or } \min \quad \hat{z}_{n^z-1-t},$$

and the following constraints should be added,

$$\hat{z}_{n^z-1-t'} = \hat{z}^*_{n^z-1-t'} \qquad\qquad t' \in J^z : \ t' < t,$$

where $\hat{z}^*_{n^z-1-t'}$ is the optimal value of bit $\hat{z}_{n^z-1-t'}$ obtained in each iteration t'. We now make a few observations.

Observation 6.1. In the Bitwise search mechanism, in each iteration, one feasible solution will be naturally found for the multiplicative program. Therefore, a global primal bound can be computed for a multiplicative program after each iteration.

To understand Observation 6.1, consider the Altogether reformulation and let

$$\hat{\boldsymbol{z}}^t := (z_0^t, \ldots, z_{n^z-1}^t)$$

be the value of all bits obtained in iteration $t \in J^z$. Since at iteration $t \in J^z$, the optimal value of bit $\hat{z}_{n^z-1-t}$ will be obtained, we have $\hat{z}^*_{n^z-1-t} = \hat{z}^t_{n^z-1-t}$. Moreover, since $\hat{\boldsymbol{z}}^t$ is feasible

$\sum_{i \in J^z} 2^i \hat{z}_i^t$ is a global primal bound for the optimal objective value of the multiplicative program.

Observation 6.2. In the Bitwise search mechanism, in each iteration, a global dual bound can be obtained by setting the value of not-yet-explored bits to their *ideal* values.

To understand Observation 6.2, note that for mMPs, the ideal value, denoted by v^*, for any arbitrary bit is zero, i.e., $v^* = 0$, as the problem is in the form of minimization (but that may not be feasible). Similarly, for MMPs, the ideal value for any arbitrary bit is one, i.e., $v^* = 1$, as the problem is in the form of maximization (but that, too, may not be feasible). Hence, for the Altogether reformulation, after obtaining $\hat{\boldsymbol{z}}^t$ in iteration $t \in J^z$, we know that

$$\sum_{t'=0}^{t} 2^{n^z-1-t'} \hat{z}^t_{n^z-1-t'} + \sum_{t'=t+1}^{n^z-1} 2^{n^z-1-t'} v^*$$

is a global dual bound for the optimal objective value of the multiplicative program.

Observation 6.3. In the Altogether reformulation, after obtaining $\hat{\boldsymbol{z}}^t$ in iteration $t \in J^z$, if $\hat{z}^t_{n^z-1-(t+1)} = v^*$, then iteration $t+1$ can be skipped as $\hat{\boldsymbol{z}}^t$ is optimal for iteration $t+1$, too, i.e., we can set $\hat{\boldsymbol{z}}^{t+1} = \hat{\boldsymbol{z}}^t$.

Due to the importance of Observation 6.3, when implementing the Bitwise search mechanism, we assume that the observation is employed. In other words, we assume that in the basic version of the Bitwise search mechanism, Observation 6.3 is included for removing redundant calculations and reducing the number of optimization problems needed to be solved. In addition to the basic version of Bitwise, in this study, we consider two additional variations of the Bitwise search mechanism which we will present next.

6.3.2.2.1 Bitwise + Full Index Set Cut (Bitwise + F-cut)

This is the basic version of the Bitwise search mechanism with one additional cut (referred to as F-cut) that needs to be added if an optimization problem is solved at iteration t. The *full index set* at iteration t is the set of all indices of not-yet-explored bits whose values in the solution found in iteration $t-1$ are not equal to v^*, i.e., the ideal value. Note that because of Observations 6.1 and 6.3, the solution found at iteration $t-1$ corresponds to the best global primal bound known at the beginning of iteration t. For the Altogether reformulation, the full index set at iteration t, denoted by $\mathcal{F}^t$, can be defined as

$$\mathcal{F}^t = \Big\{ i \in \{0, \ldots, n^z - 1 - t\} : \ \hat{z}_i^{t-1} \neq v^* \Big\}.$$

Observation 6.4. If the solution found at iteration $t-1$ is not globally optimal for the multiplicative program, then at least one of the bits with indices in $\mathcal{F}^t$ should take its ideal value.

Based on Observation 6.4, for the Altogether reformulation, the F-cut can be defined as

$$v^* \sum_{i \in F^t} \hat{z}_i + (1 - v^*) \sum_{i \in F^t} (1 - \hat{z}_i) \geqslant 1.$$

Observation 6.5. If after adding the F-cut at iteration t, the optimization problem that needs to be solved at iteration t becomes infeasible, then the solution found at iteration $t-1$ is globally optimal and the Bitwise search can terminate immediately.

6.3.2.2.2 Bitwise + Partial Index Set Cut (Bitwise + P-cut)

This variation is similar to Bitwise + F-cut. The main difference is that F-cut will be replaced by a different cut, referred to as P-cut. At the beginning of iteration t, let α^t be the index of the most significant not-yet-explored bit whose value in the solution found in iteration $t-1$ is equal to v^*, i.e., the ideal value. In other words, for the Altogether reformulation α^t can be defined as

$$\alpha^t = \max \Big\{ i \in \{0, \ldots, n^z - 1 - t\} : \ \hat{z}_i^{t-1} = v^* \Big\}.$$

The *partial index set*, denoted by $\widetilde{\mathcal{F}}^t$, is the set of all elements in $\mathcal{F}^t$ that are greater than α^t, i.e.,

$$\widetilde{\mathcal{F}}^t = \big\{ i \in \mathcal{F}^t : i > \alpha^t \big\}.$$

Using $\widetilde{\mathcal{F}}^t$, for the Altogether reformulation, the P-cut can be defined as

$$v^* \sum_{i \in \widetilde{\mathcal{F}}^t} \hat{z}_i + (1 - v^*) \sum_{i \in \widetilde{\mathcal{F}}^t} (1 - \hat{z}_i) \geqslant 1.$$

Note that $\widetilde{\mathcal{F}}^t$ is a subset of sequential most significant not-yet-explored bits whose values are not ideal in the solution obtained in iteration $t-1$. Hence, the P-cut is designed to ensure that at least one of these bits will take its ideal value in iteration t if possible. However, if it is not possible, then we have the following observation that helps us skip some iterations.

Observation 6.6. If after adding the P-cut at iteration t, the optimization problem that needs to be solved at iteration t becomes infeasible, then the optimal value of the bits with indices in $\widetilde{\mathcal{F}}^t$ cannot be ideal. In other words, their values will surely be $1 - v^*$. This implies that for the Altogether reformulation, we can set $\hat{\boldsymbol{z}}^{t'} = \hat{\boldsymbol{z}}^{t-1}$ for all $t' \in \{t, t+1, \ldots, t + |\widetilde{\mathcal{F}}^t| - 1\}$, and all iterations $\{t, t+1, \ldots, t + |\widetilde{\mathcal{F}}^t| - 1\}$ can be skipped.

6.3.3 Warm-start Enhancements for Minimum Multiplicative Problems

Similar to the previous section, again without loss of generality, we explain our enhancements only in the context of the Altogether reformulation. However, the same enhancements can be applied to the Nested reformulation by simply adding the subscript p to some notations used in this section. Specifically, any instance of n^z and $\hat{z}^w_i$ should be replaced by n^z_p and $\hat{z}^w_{p,i}$, respectively. With this in mind, suppose that before starting a search mechanism (One-shot or Bitwise), a feasible solution is known for a multiplicative program that can be used for warm-starting. We denote the objective bit values of this feasible solution by $\hat{\boldsymbol{z}}^w := (\hat{z}^w_0, \ldots, \hat{z}^w_{n^z-1})$ for the Altogether reformulation. Moreover, we denote the most significant bit in $\hat{\boldsymbol{z}}^w$ that has not taken its ideal value by β^w, i.e.,

$$\beta^w := \max\left\{i \in \{0, \dots, n^z - 1\} : \ \hat{z}^w_{n^z-1} \neq v^*\right\}$$

for the Altogether reformulation.

Observation 6.7. In the Altogether reformulation, the optimal values of bits with indices $\{\beta^w + 1, \beta^w + 2, \dots, n^z - 1\}$ are equal to the ideal value, i.e., $\hat{z}_i = v^*$ for all $i \in \{\beta^w + 1, \beta^w + 2, \dots, n^z - 1\}$.

Observation 6.7 is important as it suggests that the optimal value of some of the most significant bits can be determined in advance if a feasible solution is known in advance. In practice, this observation is more likely to be effective for mMPs. This is because the maximum number of bits required to represent the optimal objective value of a multiplicative program, i.e., n^z (for the Altogether formulation), is determined based on the product of the upper bounds on y_1,..., y_p. Therefore, since mMPs are in the form of minimization, many of the most significant bits naturally take the optimal value of zero. We hence suggest to employ Observation 6.7 only for mMPs. Next, we present two simple ways of computing a good initial feasible solution for mMPs.

6.3.3.1 Min-Min Appraoch

Our first proposed approach minimizes the minimum of $y_1, \dots, y_p$ over the feasible set. The following (mixed) integer linear program needs to be solved:

$$\min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{\lambda},\theta} \left\{\theta : (6.3),\ (6.4),\ (6.5),\ \theta \geqslant y_i - M\lambda_i \quad i \in I,\ \sum_{i \in I} \lambda_i = p - 1,\ \lambda_i \in \{0, 1\} \quad i \in I\right\},$$

where M is a sufficiently large number. We denote the vector of optimal values for the y-variables by $\boldsymbol{y}^w$. The binary-encoded version of $\prod_{i \in I} y^w_i$ generates $\hat{\boldsymbol{z}}^w$.

6.3.3.2 Indirect Min-Min Approach

In our second approach, we first minimize each y_i for $i \in I$ separately and then choose the solution of the problem that results in the smallest objective value for mMP. In other words, for each $i \in I$, we solve

$$\arg\min_{\boldsymbol{x},\boldsymbol{y}} \big\{ y_i : (6.3),\ (6.4),\ (6.5) \big\},$$

and denote the vector of optimal values for the y-variables by $\boldsymbol{y}^{w,i}$. The binary-encoded version of

$$\min \left\{ \prod_{i \in I} y_i^{w,1}, \dots, \prod_{i \in I} y_i^{w,p} \right\}$$

generates $\hat{\boldsymbol{z}}^w$.

6.3.4 Continuous Cases

In all previous cases, we assumed that all y-variables are integer. In this section, we explain how our approach can be used for approximating multiplicative programs with $p_C \neq 0$ (with any desirable level of precision). Suppose that there exists an $i \in I$ such that y_i can only take fractional values. Suppose further that users are interested in only $\beta \in \mathbb{N}_0$ digits after the decimal point of the value of y_i. The following two steps should be applied in order to modify the Nested or the Altogether reformulation:

- Step 1: We first find the equation corresponding to y_i from Constraint (6.3). We then multiply its right-hand side by 10^{β}. Note that to avoid running into numerical issues and removing any solutions, we keep the declaration of y_i as continuous variables, i.e., we do not change it to integer. As an aside, we note that Constraint (6.3) appears in both the Nested and the Altogether reformulation. Hence, both reformulations will be modified in this step.

- Step 2: We change the equation corresponding to y_i in Constraints (6.12), (6.13), and (6.26) from equality to inequality. Specifically, we change '$=$' to '$\geqslant$' for the maximization instances. Moreover, we change '$=$' to '$\leqslant$' for the minimization instances. These inequalities ensure that the optimal binary-encoded value of y_i represents $\lfloor y_i \rfloor$ and $\lceil y_i \rceil$ in MMPs and mMPs, respectively. In other words, our approximation technique is a lower approximation for MMPs and an upper approximation for mMPs.

6.4 Extreme Examples

As mentioned earlier, the Bitwise search mechanism is the only strategy (in this paper) that can truly resolve the curse of multiplication. Moreover, the Nested reformulation keeps the size of the problem polynomially bounded while the Altogether reformulation increases the size of the problem exponentially (as p increases). To solve instances with large values of p, the only suitable approach is to employ the Nested reformulation when being combined with the Bitwise search mechanism. In order to testify this claim, in this section, we compare the performance of our approach, i.e., Nested reformulation + Bitwise search mechanism, with an off-the-shelf solver on two instances where the curse of multiplication is likely to arise.

Both instances have 20 binary variables, i.e., $n_C = 0$ and $n_I = 20$, and 10 constraints in the space of x-variables, i.e., $m = 10$. However, one instance has $p = 15$ and the other has $p = 20$. Interested readers may refer to Appendix E2 to find the fully defined instances. As an aside, we note that although these instances look small, simply generating the Altogether reformulation can consume a significant amount of time and memory space due to $p \in \{15, 20\}$. In fact we could not load the Altogether reformulations of either one of these two instances in a computing platform with 128GB RAM. However, the Nested reformulations require a negligible amount of time and memory space and that is why we have used it in this section.

Table 6.3: Results for the instance with $p = 15$

	Min		Max	
	Objective value	Time (s)	Objective value	Time (s)
Bitwise	37,881,049,842,155,520	669.21	13,426,599,939,480,000,000	2107.71
GRB	131,262,150,868,992,000	0.07	2,456,521,675,442,388,480	0.05

We solve each instance both in the form of minimization and maximization using our approach, i.e., Nested reformulation + Bitwise search mechanism. Additionally, we solve each instance using a powerful commercial off-the-shelf solver, Gurobi version 9.0.2 (which we refer to as GRB), to conduct comparisons. As previously mentioned in Section 6.2, in order to solve multiplicative programs using GRB, an MMP can be reformulated as a (mixed integer) SOCP and an mMP can be reformulated as a nonconvex optimization program, specifically, a nonconvex quadratically constrained quadratic program. Hence, we employ GRB SOCP for the maximization form and GRB Nonconvex for the minimization form. For consistency, when implementing our algorithm, we also set GRB as the default solver for solving mixed integer linear programs arising during the course of our algorithm. We impose a time limit of 259,200 seconds (3 days) and set the optimality gap tolerance to zero for all solution approaches. The results are shown in Tables 6.3 and 6.4 for $p = 15$ and $p = 20$, respectively.

Table 6.4: Results for the instance with $p = 20$

	Min		Max	
	Objective value	Time (s)	Objective value	Time (s)
Bitwise	322,275,494,202,543,737,864,192	259,200.00	65,290,849,092,115,078,053,888,000	259,200.00
GRB	*Infeasible*	0.00	–	259,200.00

In Table 6.3, we observe that GRB claims optimality within a fraction of a second. However, obviously, the objective values reported are far from the optimal values reported by the Bitwise approach. This is an indication that GRB has faced numerical issues. In Table 6.4, we observe that GRB Nonconvex (for minimization) immediately reports infeasibility and GRB SOCP (for maximization) does not report any feasible solution within the time limit. However, our Bitwise approach found feasible solutions for both cases reporting $4 \times 10^{-7}\%$ and 98.6% optimality gap for minimization and maximization, respectively. We note that the 98.6% optimality gap reported for the maximization case is due to the magnitude of the objective value.

6.5 Computational Study

In this computational study, we compare the performance of our proposed algorithms with GRB SOCP and GRB Nonconvex when solving MMPs and mMPs for small values of p, respectively. All solution methods as well as our instance generator are implemented in C++ (the source files are available at https://github.com/paymanghasemi/Multiplicative-Programs-by-Binary-encoding-the-Multiplication-Operation and the '.lp' format of the instances are available at https://usf.box.com/s/1u6xesylwufybxj8fjwyefslcxw77nyt). For consistency, when implementing our algorithms, we also set GRB as the default solver for solving mixed integer linear programs arising during the course of our algorithms. All 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 7.0 operating system, using a single thread. We set the optimality gap tolerance of GRB to zero in all experiments. Moreover, a time limit of 3,600 seconds (1 hour) is imposed for solving each instance using each solution method.

We generate 900 instances for this computational study through the procedure described in Appendix E3. Specifically, we generate 300 binary instances that can be used in both maximization and minimization forms, 300 continuous maximization instances, and 300 continuous minimization instances. Our computational study is based on 5,400 runs where each run means solving an instance by a specific solution method. These 5,400 runs are divided (not equally) into four experiments:

- *Experiment 1:* The first experiment focuses on identifying the best search mechanism. We conduct this experiment on only instances with $p = 2$ as the Altogether and Nested reformulations are the same for such instances. Since our solution methods are only approximate algorithms for continuous instances, we only focus on pure binary instances for the first experiment. We show that One-shot is the fastest search mechanism. Hence, in the remaining experimental settings, One-shot is set as the default search mechanism for our solution approaches.

- *Experiment 2:* In the second experiment, we focus on binary instances in minimization form with $p \in \{2, 3\}$ to identify whether the proposed warm-start techniques are effective and, if so, which one performs the best. Note that we do not introduce any enhancements for the maximization instances (in this paper) and that is why the focus of this experiment is only on minimization instances. In this experiment, we use the Nested reformulation (combined with the One-shot search mechanism). Overall, based on the results of the second experiment, we show that 'Indirect Min-Min' performs best. Therefore, in the remaining experiments, 'Indirect Min-Min' is the default enhancement technique when solving minimization instances.

- *Experiment 3:* In the third experiment, we compare the performance of the Nested and Altogether reformulations (when being combined with the One-shot search mechanism and the Indirect Min-Min enhancement) with the performance of GRB on all pure binary instances with $p \in \{2, 3, 4\}$. Specifically, for pure binary maximization instances, GRB SOCP is the benchmark for comparison, and for pure binary minimization instances, GRB Nonconvex is the benchmark for comparison. We show that GRB SOCP performs best for maximization instances while for minimization instances, the Nested reformulation is the dominant approach.

- *Experiment 4:* In the fourth experiment, we repeat Experiment 3 but for all continuous instances with $p \in \{2, 3, 4\}$. The only difference is that due to the poor performance of the Altogether reformulation, we do not use it in this experiment. Moreover, as our proposed approaches are approximations for continuous instances, we consider three multipliers $\{10^0, 10^2, 10^4\}$ for transforming the continuous y-variables into integers, i.e.,

$\beta \in \{0, 2, 4\}$. We show that GRB SOCP performs best for maximization instances while for minimization instances, the Nested reformulation is the dominant approach.

In light of the above, the following are the solution approaches (in addition to GRB SOCP and GRB Nonconvex) that we use in our experiments for binary instances:

- N-O: Nested, One-shot,
- N-B: Nested, Bitwise,
- N-B+F: Nested, Bitwise + F-cut,
- N-B+P: Nested, Bitwise + P-cut,
- N-O-Imm: Nested, One-shot, Indirect Min-Min,
- N-O-mm: Nested, One-shot, Min-Min,
- A-O: Altogether, One-shot,
- A-O-Imm: Altogether, One-shot, Indirect Min-Min.

Specifically, N-O, N-B, N-B+F, and N-B+P are used in Experiment 1. Note that the Nested and Altogether reformulations are indeed the same when dealing with instances of $p = 2$. Hence, the term 'Nested' can be replaced by 'Altogether' for instances with $p = 2$. In Experiment 2, N-O, N-O-Imm, and N-O-mm are used. In Experiment 3, N-O, A-O, and GRB SOCP are used for maximization instances while N-O-Imm, A-O-Imm, and GRB Nonconvex are used for minimization instances. The following are the solution approaches (in addition to GRB SOCP and GRB Nonconvex) that we used in our experiments for continuous instances:

- N-O-M: N-O with M as multiplier from the set $\{10^0, 10^2, 10^4\}$,
- N-O-Imm-M: N-O-Imm with M as multiplier from the set $\{10^0, 10^2, 10^4\}$.

Table 6.5: High-level summary of computational results

Direction	Variables	p = 2		p = 3		p = 4	
Maximize	Binary	# Runs:	600	# Runs:	300	# Runs:	300
		# Instances:	100	# Instances:	100	# Instances:	100
		GRB SOCP	85.00%	GRB SOCP	82.00%	GRB SOCP	87.00%
		Time outs	14.00%	Time outs	18.00%	Time outs	13.00%
		N-O	1.00%				
	Continuous	# Runs:	400	# Runs:	400	# Runs:	400
		# Instances:	100	# Instances:	100	# Instances:	100
		GRB SOCP	95.00%	GRB SOCP	85.00%	GRB SOCP	92.00%
		N-O-10^0	4.00%	N-O-10^0	8.00%	Time outs	7.00%
		N-O-10^4	1.00%	Time outs	6.00%	N-O-10^4	1.00%
				N-O-10^4	1.00%		
Minimize	Binary	# Runs:	900	# Runs:	600	# Runs:	300
		# Instances:	100	# Instances:	100	# Instances:	100
		N-O-Imm	76.17%	N-O-Imm	75.00%	N-O-Imm	40.00%
		N-O	17.17%	Time outs	12.00%	Time outs	36.00%
		GRB Nonconvex	3.33%	A-O-Imm	11.00%	GRB Nonconvex	24.00%
		N-O-mm	3.33%	GRB Nonconvex	1.00%		
				N-O	1.00%		
	Continuous	# Runs:	400	# Runs:	400	# Runs:	400
		# Instances:	100	# Instances:	100	# Instances:	100
		N-O-Imm-10^0	94.00%	N-O-Imm-10^0	100.00%	N-O-Imm-10^0	89.00%
		N-O-Imm-10^2	3.00%			GRB Nonconvex	7.00%
		GRB Nonconvex	2.00%			N-O-Imm-10^2	3.00%
		N-O-Imm-10^4	1.00%			Time outs	1.00%

Specifically, in Experiment 4, N-O-10^0, N-O-10^2, N-O-10^4, and GRB SOCP are used for maximization instances while N-O-Imm-10^0, N-O-Imm-10^2, N-O-Imm-10^4, and GRB Nonconvex are used for minimization instances.

Table 6.5 shows a high-level summary of the computational results of all 5,400 runs. It is worth mentioning that the result tree shown in Figure 6.1 (given in the introduction) is generated based on Table 6.5. In this table, we classify the problem variation by the *direction* (maximize, minimize), the *variables* (binary, continuous), and $p \in \{2, 3, 4\}$. Hence, in total there are $2 \times 2 \times 3 = 12$ problem variations. For each problem variation, we provide the number of unique runs and instances in Table 6.5. Below that, we list the solution methods and their share of instances for which they had the shortest solution time. For some instances, there may be a tie between two or more solution methods, which explains the fractional percentages. For example, for the maximization, binary, and $p = 2$ case, GRB SOCP had the shortest solution time for 85% of instances, followed by 14% time outs and 1% One-shot. Solution methods that never had the shortest solution time for any instance (i.e., 0%) are omitted. It is immediately clear that GRB SOCP performs best for maximization problems. For minimization problems, however, the solution methods introduced in this paper outperform the off-the-shelf benchmarks. Specifically, for binary problems, N-O-Imm and N-O perform best for $p = 2$. For $p = 3$, the top performers are N-O-Imm and A-O-Imm. It is only for $p = 4$ that the instances either time out or can only be solved by GRB Nonconvex or N-O-Imm. For continuous problems, N-O-Imm-10^0 approximations dominate the other approaches.

To see more details of Experiments 1–4, interested readers may refer to Appendix E4. Moreover, there is a 185-page supplementary PDF document (along with its corresponding CSV file) available at https://github.com/paymanghasemi/Multiplicative-Programs-by-Binary-encoding-the-Multiplication-Operation for details on every one of the 5,400 runs in this study.

Chapter 7: Conclusions and Future Research Directions

7.1 Conclusions of Chapter 2

We developed a bi-objective linear programming based branch-and-bound algorithm for solving a class of mixed integer linear maximum multiplicative programs (with a bi-linear objective function). This class of optimization problems has only one objective function and it can be solved directly by a commercial mixed integer second-order cone programming solver. However, it was shown that the proposed branch-and-bound algorithm outperforms such a solver by a factor of around 2 on average. Using a computational study, different branching and node selecting strategies as well as enhancement techniques were explored. It was shown that the most infeasible branching and best-bound search strategies perform the best for the proposed branch-and-bound algorithm. However, enhancement techniques were only useful for mixed binary instances. One drawback of the proposed method is that it can only be applied to mixed integer linear maximum multiplicative programs in which its objective function involves a bi-linear term. Therefore, developing a multi-objective optimization based algorithm for cases where the objective function involves a multi-linear term can be a future research direction.

7.2 Conclusions of Chapter 3

We developed a multi-objective mixed binary linear programming based algorithm for solving MIL-MMPs. This class of optimization problems has only one objective function and can be reformulated as mixed integer second-order cone programs. The reformulation can be directly solved by a commercial solver such as CPLEX. Using a detailed computational study, we demonstrated that our proposed algorithm significantly outperforms such a solver.

We also showed that our algorithm outperforms a recent algorithm for solving instances of MIL-MMPs with $p = 2$. Finally, we showed that even by linearizing the objective function and solving the resulted (mixed) integer linear program by a commercial solver, the solution time will be significantly larger than the one obtained by our proposed approach.

As for the future research guideline, the following shortcomings of the current paper can be of interest. As mentioned in Section 3.4, a major limitation of our proposed algorithm is its natural instability when solving MIL-MMPs with large values of p. The optimal objective value of MIL-MMPs tends to grow exponentially as p increases. Therefore, not surprisingly, existing solution approaches (including the one proposed in this study) do not guarantee to not run into numerical instabilities when dealing with such instances. So, studying algorithms that are theoretically guaranteed to not run into objective-value-caused numerical instabilities can be an interesting future research direction. Another limitation of our proposed algorithm is that it requires converting all integer decision variables to binaries which in turn increases the overall size of a formulation. Specifically, in our proposed approach, any integer decision variable with an upper bound equal to u needs to be replaced by exactly $\lfloor \log_2 u \rfloor + 1$ binary variables. Moreover, to complete the transformation, one additional constraint needs to be added to the formulation for each integer variable. Such a transformation is expensive but makes the process of adding no-good constraints, which are essential for the correctness of our proposed approach, more conveniently. Therefore, studying algorithms that do not require adding no-good constraints and/or the transformation of integer variables can be another interesting future research direction of this study as such algorithms can possibly have computational advantages over the one proposed in this paper.

7.3 Conclusions of Chapter 4

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 the 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 the 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 [1]. Their proposed approach is motivated by this simple observation that for solving any mixed integer optimization problem four steps can be done iteratively: (1) Finding a new feasible solution for the optimization model and then updating the best solution found so far accordingly. (2) Fixing the values of the integer decisions by setting them to their corresponding values in the feasible solution, the so-called integer support vector, in order to generate a continuous optimization problem. (3) Solving the continuous optimization problem to obtain the best feasible solution for the integer support vector used and then updating the best solution found so far accordingly. (4) Adding a no-good constraint to the model for removing the integer support vector permanently from the search and then returning to Step 1.

In light of the above, in order to customize our proposed algorithms for solving MIL-GMMPs, 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 $\text{WSO}(\boldsymbol{w}, \textsc{Pool}, \boldsymbol{l})$ 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 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 a 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 is 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 [125, 126, 54, 55] and

[114]). 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 [55]. 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.

7.4 Conclusions of Chapter 5

In this research, we proposed a methodology that, for the first time to the best of our knowledge, models a fluency map problem as a game. Specifically, we proposed a bargaining game where players/organs start bargaining from their worst expected DVH and try to get a final solution closer to their ideal plan. During this process, we redefined the fluency map optimization such that, instead of the typical penalty functions, we have new objective functions referred to as preference functions. The advent of preference functions has several advantages. First, the preference values of all organs are unitless and are expected to be in the range of zero to one, which makes the objective values comparable to each other. Second, the values of the new functions are meaningful as they represent the similarity percentage of each player's/organ's plan to its ideal plan.

Further, to assure the efficiency and mathematically provable fairness of the final solution of our proposed game, we utilized the concept of Nash Social Welfare in our proposed methodology. The use of Nash Social Welfare enabled us to introduce a new control lever for the fluency map optimization, the so-called negotiation powers. These powers control the similarity rate of each player/organ to their ideal plan in the final solution and provide the flexibility of putting more emphasis on an organ by increasing its negotiation power.

We note that we tried to provide the most straightforward form of our methodology in this paper, and an extensive future research is needed to improve the proposed methodology to its full capacity. Specifically, as a guideline for future research directions, we have created Figure 7.1 which is a general framework for extending our methodology. In Figure 7.1, the potential blocks to improve the final plan are presented in gray, and the dashed lines indicate the potential loops that can be added to the methodology to construct an automated algorithm. Next, we discuss the future research directions based on this figure.

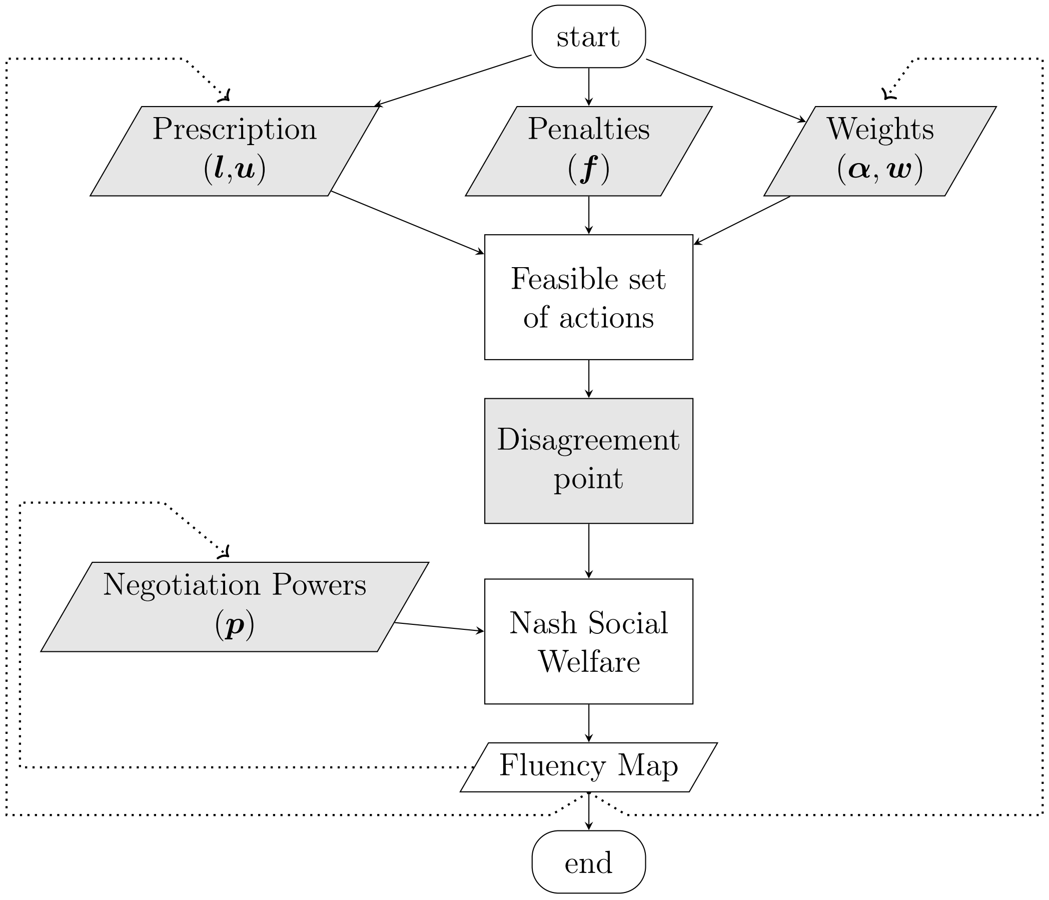

start
Prescription ($\boldsymbol{l}$,$\boldsymbol{u}$)
Penalties ($\boldsymbol{f}$)
Weights ($\boldsymbol{\alpha}, \boldsymbol{w}$)
Feasible set of actions
Disagreement point
Negotiation Powers ($\boldsymbol{p}$)
Nash Social Welfare
Fluency Map
end

Figure 7.1: A general framework for extending our proposed methodology

- *Penalties (f)*: Penalty functions play the role of utility functions in our methodology and are critical in defining the feasible set of actions and finding the disagreement point. As a result, the quality of the final plan highly depends on the choice of the penalty functions. Although we used the quadratic function as our main penalty type and tried to get closer to the ideal solution defined by this function, one can use different concepts as the initial criterion. There are some linear and non-linear objective functions that can be utilized in our proposed approach [74, 71, 75, 127, 128, 129, 130], each of which has its own pros and cons. So, a future research guideline is to study the impact of different penalty types in the final plan of our methodology.

- *Prescription ($\boldsymbol{l}$,$\boldsymbol{u}$)*: In our methodology, the goal is to get closer to the ideal plan while distancing from the worst plan, both of which are defined based on the initial prescription. If a prescription with higher alignment with the status of the patience is provided, less diverged ideal and worst cases can be defined which can impact the final plan drastically. One approach for so doing is to iteratively update the prescribed dose levels, which can be done by including the dashed line entering to the *Prescription* block in Figure 7.1. Hence, a guideline for the future research is creating an automated approach for updating the prescriptions to improve the final plan. A similar research has been done by [76], where instead of the prescription, a threshold is introduced to the model and updated iteratively.

- *Initial Weights ($\boldsymbol{\alpha}, \boldsymbol{w}$)*: In Remark 5.1, we mentioned that how manipulating the vector of importance weights can affect the final plan. Therefore, a smart choice of the initial importance weights, similar to ones proposed by [131] and [132], can be a good candidate to improve the final plan. In addition, further improvement of the importance weights is possible by including the dashed line entering to the *Weights* block in Figure 7.1 and using the methods that iteratively update the importance weights [78].

- *Disagreement points*: In Section 5.3.3, we proposed a simple approach to find the reference points for our game, i.e., the ideal and worst penalty values. However, practitioners can use their preferred values as the reference points. For instance, the penalty values resulted by the quadratic fluency map optimization, i.e., Problem (5.9), can be implemented as the disagreement point. By so doing, the model will try to get farther from the result of Problem (5.9) rather than the real worst expected values. Another approach can be estimating the ideal and worst values [133] which will reduce the computational efforts of the methodology.

- *Negotiation powers (**p**)*: As mentioned in Section 5.3.4, the negotiation powers are new control levers introduced for the first time for fluency map optimization. Having fixed initial prescriptions and importance weights, the negotiation powers provide the ability to improve the final plan. Particularly, by developing an update role for negotiation powers and including the dashed line entering the *Negotiation powers* block in Figure 7.1, an automated algorithm can be constructed to improve the final plan. Therefore, a future research guideline is developing an automated algorithm for updating the negotiation powers.

7.5 Conclusions of Chapter 6

In this paper, we addressed the question of how the curse of multiplication can be resolved when solving multiplicative programs. Specifically, we showed how the multiplication operation itself can be binary-encoded as an integer linear program following the procedure that a computer uses internally. We developed two types of binary-encoded reformulations for a multiplicative program: the Nested reformulation and the Altogether reformulation. Assuming that a multiplicative program has a compact (i.e., polynomially bounded) size, we proved that the Nested reformulation will also have a compact size. However, that is not the case for the Altogether reformulation. We also introduced two search mechanisms, One-shot and Bitwise, with several enhancement techniques for solving the proposed reformulations. One-shot is suitable for multiplicative programs with small values of p, while Bitwise is suitable for multiplicative programs with large values of p. Therefore, the combination of the Nested reformulation and the Bitwise search mechanism is the first approach (to the best of our knowledge) that can fully resolve the curse of multiplication in multiplicative programs.

Although our focus has been mainly on resolving the curse of multiplication, surprisingly, we observed that the Nested reformulation combined with the One-shot search mechanism completely outperforms standard solvers on minimization instances, regardless of the value of p. However, for maximization instances with small values of p, standard solvers perform best. This difference can be explained by the fact that maximization instances are different from minimization instances in nature, i.e., continuous mMPs are NP-hard while continuous MMPs are polynomially solvable.

We hope that the simplicity of our proposed solution approaches and their promising results encourage more researchers to study multiplicative programs. Specifically, many different reformulations and/or search mechanisms can be developed by combining ideas behind those presented in this paper. Therefore, studying such reformulations and search mechanisms both theoretically and computationally could be valuable.

References

[1] Payman Ghasemi Saghand and Hadi Charkhgard. A criterion space search algorithm for mixed integer linear maximum multiplicative programs: a multiobjective optimization approach. *International Transactions in Operational Research*, 2021. Available online.

[2] Mamoru Kaneko and Kenjiro Nakamura. The nash social welfare function. *Econometrica*, 47(2):423–435, 1979.

[3] Hadi Charkhgard, Martin Savelsbergh, and Masoud Talebian. 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, 2018. `https://doi.org/10.1016/j.cor.2017.07.015`.

[4] Payman Ghasemi Saghand, Hadi Charkhgard, and Changhyun Kwon. 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, 2019.

[5] J F Nash. The bargaining problem. *Econometrica*, 18:155–162, 1950.

[6] J F Nash. Two-person cooperative games. *Econometrica*, 21:128–140, 1953.

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

[8] Natashia Boland, Hadi Charkhgard, and Martin Savelsbergh. 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, 2017.

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

[10] Harold P Benson. Optimization over the efficient set. *Journal of Mathematical Analysis and Applications*, 98(2):562–580, 1984.

[11] Roberto Serrano. Fifty years of the Nash program 1953-2003. *Investigaciones Economicas*, pages 219–258, 2005.

[12] Ioannis Caragiannis, David Kurokawa, Hervé Moulin, Ariel D. Procaccia, Nisarg Shah, and Junxing Wang. The unreasonable fairness of maximum nash welfare. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, EC '16, pages 305–322, New York, NY, USA, 2016. ACM.

[13] Dongmo Zhang. A logic-based axiomatic model of bargaining. *Artificial Intelligence*, 174(16):1307 – 1322, 2010.

[14] John P. Conley and Simon Wilkie. The bargaining problem without convexity: Extending the egalitarian and kalai-smorodinsky solutions. *Economics Letters*, 36(4):365 – 369, 1991.

[15] Deeparnab Chakrabarty, Nikhil Devanur, and Vijay V Vazirani. New results on rationality and strongly polynomial time solvability in Eisenberg-Gale markets. In *Internet and Network Economics*, volume 4286 of *Lecture Notes in Computer Science*, pages 239–250. Springer Berlin Heidelberg, 2006.

[16] Edmund Eisenberg and David Gale. Consensus of subjective probabilities: The parimutuel method. *The Annals of Mathematical Statistics*, 30(1):165–168, 1959.

[17] Kamal Jain and Vijay V Vazirani. Eisenberg-Gale markets: Algorithms and structural properties. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 364–373, New York, NY, USA, 2007. ACM.

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

[19] Thomas Stidsen, Kim Allan Andersen, and Bernd Dammann. A branch and bound algorithm for a class of biobjective mixed integer programs. *Management Science*, 60(4):1009–1032, 2014.

[20] Özgür Özpeynirci and Murat Köksalan. An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs. *Management Science*, 56(12):2302–2315, 2010.

[21] Selcen (Pamuk) Phelps and Murat Köksalan. An interactive evolutionary metaheuristic for multiobjective combinatorial optimization. *Management Science*, 49(12):1726–1738, 2003.

[22] Thomas Erlebach, Hans Kellerer, and Ulrich Pferschy. Approximating multiobjective knapsack problems. *Management Science*, 48(12):1603–1612, 2002.

[23] Michael P. Johnson and Arthur P. Hurter. Decision support for a housing mobility program using a multiobjective optimization model. *Management Science*, 46(12):1569–1584, 2000.

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

[25] Alexander Engau and Margaret M. Wiecek. Interactive coordination of objective decompositions in multiobjective programming. *Management Science*, 54(7):1350–1363, 2008.

[26] Jyrki Wallenius, James S. Dyer, Peter C. Fishburn, Ralph E. Steuer, Stanley Zionts, and Kalyanmoy Deb. Multiple criteria decision making, multiattribute utility theory: Recent accomplishments and what lies ahead. *Management Science*, 54(7):1336–1349, 2008.

[27] S.S. Rao. Game theory approach for multiobjective structural optimization. *Computers & Structures*, 25(1):119 – 127, 1987.

[28] Hadi Charkhgard, Kimia Keshanian, Rasul Esmaeilbeigi, and Parisa Charkhgard. The magic of Nash social welfare in optimization: Do not sum, just multiply. *Preprint*, 2020.

[29] Emily Nicholson and Hugh P. Possingham. Objectives for multiple-species conservation planning. *Conservation Biology*, 20(3):871–881, 2006.

[30] Zulqarnain Haider, Hadi Charkhgard, and Changhyun Kwon. A robust optimization approach for solving problems in conservation planning. *Ecological Modelling*, 368:288 – 297, 2018.

[31] Alvaro Sierra-Altamiranda, Hadi Charkhgard, Mitchell Eaton, Julien Martin, Simeon Yurek, and Bradley J. Udell. Spatial conservation planning under uncertainty using modern portfolio theory and nash bargaining solution. *Ecological Modelling*, 423:109016, 2020.

[32] David E Calkin, Claire A Montgomery, Nathan H Schumaker, Stephen Polasky, Jeffrey L Arthur, and Darek J Nalle. Developing a production possibility set of wildlife species persistence and timber harvest value. *Canadian Journal of Forest Research*, 32(8):1329–1342, 2002.

[33] Emily Nicholson and Hugh P. Possingham. Objectives for multiple-species conservation planning. *Conservation Biology*, 20(3):871–881, 2006.

[34] Paul H. Williams and Miguel B. Araújo. Apples, oranges, and probabilities: Integrating multiple factors into biodiversity conservation with consistency. *Environmental Modeling & Assessment*, 7(2):139–151, Jun 2002.

[35] David W. Coit. Cold-standby redundancy optimization for nonrepairable systems. *IIE Transactions*, 33(6):471–478, Jun 2001.

[36] Mostafa Abouei Ardakan and Ali Zeinal Hamadani. Reliability optimization of series–parallel systems with mixed redundancy strategy in subsystems. *Reliability Engineering & System Safety*, 130:132 – 139, 2014.

[37] Mohammad Feizabadi and Abdolhamid Eshraghniaye Jahromi. A new model for reliability optimization of series-parallel systems with non-homogeneous components. *Reliability Engineering & System Safety*, 157:101 – 112, 2017.

[38] David A. Hensher. Sequential and full information maximum likelihood estimation of a nested logit model. *The Review of Economics and Statistics*, 68(4):657–667, 1986.

[39] Anna Fernandez-Antolin, Virginie Lurkin, Matthieu de Lapparent, and Michel Bierlaire. Discrete-continuous maximum likelihood for the estimation of nested logit models. *16th Swiss Transport Research Conference, Ascona, Switzerland, 17-19 May*, 2017.

[40] Pierre Bonami, Jon Lee, Sven Leyffer, and Andreas Wächter. More branch-and-bound experiments in convex nonlinear integer programming. *Preprint ANL/MCS-P1949-0911, Argonne National Laboratory, Mathematics and Computer Science Division*, 2011.

[41] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[42] Michel Bénichou, Jean-Michel Gauthier, Paul Girodet, Gerard Hentges, Gerard Ribière, and O Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1(1):76–94, 1971.

[43] Alexander Martin. Integer programs with block structure, 1999.

[44] ILOG ILOG. Cplex 9.0 reference manual. *ILOG CPLEX Division*, 2003.

[45] J-P Goux and Sven Leyffer. Solving large minlps on computational grids. *Optimization and Engineering*, 3(3):327–346, 2002.

[46] Pietro Belotti, Christian Kirches, Sven Leyffer, Jeff Linderoth, James Luedtke, and Ashutosh Mahajan. Mixed-integer nonlinear optimization. *Acta Numerica*, 22:1–131, 2013.

[47] John D. C. Little, Katta G. Murty, Dura W. Sweeney, and Caroline Karel. An algorithm for the traveling salesman problem. *Operations Research*, 11(6):972–989, 1963.

[48] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[49] M Ehrgott. *Multicriteria optimization.* Springer, New York, second edition, 2005.

[50] Y P Aneja and K P K Nair. Bicriteria transportation problem. *Management Science*, 27:73–78, 1979.

[51] Aharon Ben-Tal and Arkadi Nemirovski. On polyhedral approximations of the second-order cone. *Mathematics of Operations Research*, 26(2):193–205, 2001.

[52] E D Dolan and J J Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

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

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

[55] Lizhen Shao and Matthias Ehrgott. Primal and dual multi-objective linear programming algorithms for linear multiplicative programmes. *Optimization*, 65(2):415–431, 2016.

[56] John Hooker. *Logic-based methods for optimization: combining optimization and constraint satisfaction*, volume 2. John Wiley & Sons, 2011.

[57] Matteo Fischetti, Fred Glover, and Andrea Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005.

[58] Natashia Boland, Hadi Charkhgard, and Martin Savelsbergh. A criterion space search algorithm for biobjective mixed integer programming: The triangle splitting method. *INFORMS Journal on Computing*, 27(4):597–618, 2015.

[59] Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical Programming*, 98(1):23–47, 2003.

[60] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[61] IBM ILOG CPLEX Optimization Studio. CPLEX Parameters Reference, 2016.

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

[63] M Grötschel, L Lovasz, and A Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, Berlin, 1988.

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

[65] J. Michael Steele. *The Cauchy-Schwarz Master Class: An Introduction to the Art of Mathematical Inequalities*. Cambridge University Press, 2004.

[66] Matthias Ehrgott, Çiğdem Güler, Horst W Hamacher, and Lizhen Shao. Mathematical optimization in intensity modulated radiation therapy. *Annals of Operations Research*, 175(1):309–365, 2010.

[67] F. Bartolozzi, A. de Gaetano, E. Di Lena, S. Marino, L. Nieddu, and G. Patrizi. Operational research techniques in medical treatment and diagnosis: A review. *European Journal of Operational Research*, 121(3):435 – 466, 2000.

[68] Sebastiaan Breedveld and Ben Heijmen. Data for trots–the radiotherapy optimisation test set. *Data in brief*, 12:143–149, 2017.

[69] David Craft, Mark Bangert, Troy Long, Dávid Papp, and Jan Unkelbach. Shared data for intensity modulated radiation therapy (IMRT) optimization research: the cort dataset. *GigaScience*, 3(1):37, 2014.

[70] Matthias Ehrgott, Allen Holder, and Josh Reese. Beam selection in radiotherapy design. *Linear Algebra and its Applications*, 428(5-6):1272–1312, 2008.

[71] Allen Holder. *Radiotherapy treatment design and linear programming*. Springer, 2005.

[72] Yair Censor, Adi Ben-Israel, Ying Xiao, and James M Galvin. On linear infeasibility arising in intensity-modulated radiation therapy inverse planning. *Linear algebra and its applications*, 428(5-6):1406–1420, 2008.

[73] Ying Xiao, Darek Michalski, James M Galvin, and Yair Censor. The least-intensity feasible solution for aperture-based inverse planning in radiation therapy. *Annals of Operations Research*, 119(1-4):183–203, 2003.

[74] Gino J Lim, Michael C Ferris, Stephen J Wright, David M Shepard, and Matthew A Earl. An optimization framework for conformal radiation treatment planning. *INFORMS Journal on Computing*, 19(3):366–380, 2007.

[75] Allen Holder. Partitioning multiple objective optimal solutions with applications in radiotherapy design. *Optimization and engineering*, 7(4):501–526, 2006.

[76] Troy Long, Mingli Chen, Steve Jiang, and Weiguo Lu. Threshold-driven optimization for reference-based auto-planning. *Physics in Medicine & Biology*, 63(4):04NT01, feb 2018.

[77] Hongcheng Liu, Peng Dong, and Lei Xing. A new sparse optimization scheme for simultaneous beam angle and fluence map optimization in radiotherapy planning. *Physics in Medicine & Biology*, 62(16):6428–6445, jul 2017.

[78] Masoud Zarepisheh, Troy Long, Nan Li, Zhen Tian, H. Edwin Romeijn, Xun Jia, and Steve B. Jiang. A DVH-guided IMRT optimization algorithm for automatic treatment planning and adaptive radiotherapy replanning. *Medical Physics*, 41(6Part1):061711, 2014.

[79] Chunhua Men, Xuejun Gu, Dongju Choi, Amitava Majumdar, Ziyi Zheng, Klaus Mueller, and Steve B Jiang. GPU-based ultrafast IMRT plan optimization. *Physics in Medicine and Biology*, 54(21):6565–6573, oct 2009.

[80] Sebastiaan Breedveld, Pascal R M Storchi, Marleen Keijzer, and Ben J M Heijmen. Fast, multiple optimizations of quadratic dose objective functions in IMRT. *Physics in Medicine and Biology*, 51(14):3569–3579, jul 2006.

[81] Chuan Wu, Gustavo H Olivera, Robert Jeraj, Harry Keller, and Thomas R Mackie. Treatment plan modification using voxel-based weighting factors/dose prescription. *Physics in Medicine and Biology*, 48(15):2479–2491, jul 2003.

[82] William U. Shipley, Joel E. Tepper, George R. Jr Prout, Lynn J. Verhey, Oscar A. Mendiondo, Michael Goitein, Andreas M. Koehler, and Herman D. Suit. Proton Radiation as Boost Therapy for Localized Prostatic Carcinoma. *JAMA*, 241(18):1912–1915, 05 1979.

[83] S. S. Rao and T. I. Freiheit. A Modified Game Theory Approach to Multiobjective Optimization. *Journal of Mechanical Design*, 113(3):286–291, 09 1991.

[84] Mamoru Kaneko and Kenjiro Nakamura. The nash social welfare function. *Econometrica*, 47(2):423–435, 1979.

[85] Sara Ramezani and Ulle Endriss. Nash social welfare in multiagent resource allocation. In Esther David, Enrico Gerding, David Sarne, and Onn Shehory, editors, *Agent-Mediated Electronic Commerce. Designing Trading Strategies and Mechanisms for Electronic Markets*, pages 117–131, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[86] Hadi Charkhgard, Kimia Keshanian, Rasul Esmaeilbeigi, and Parisa Charkhgard. The magic of Nash social welfare in optimization: Do not sum, just multiply. *Preprint*, 2020.

[87] Payman Ghasemi Saghand and Hadi Charkhgard. Exact solution approaches for integer linear generalized maximum multiplicative programs through the lens of multi-objective optimization. *Preprint*, 2019.

[88] S. Bechikh, L. Ben Said, and K. Ghedira. Estimating nadir point in multi-objective optimization using mobile reference points. In *IEEE Congress on Evolutionary Computation*, pages 1–9, 2010.

[89] Kalyanmoy Deb and Kaisa Miettinen. Nadir point estimation using evolutionary approaches: Better accuracy and computational speed through focused search. In *Multiple Criteria Decision Making for Sustainable Energy and Transportation Systems*, pages 339–354, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[90] Karl-Heinz Küfer, Alexander Scherrer, Michael Monz, Fernando Alonso, Hans Trinkaus, Thomas Bortfeld, and Christian Thieke. Intensity-modulated radiotherapy–a large scale multi-criteria programming problem. *OR spectrum*, 25(2):223–249, 2003.

[91] H Edwin Romeijn, James F Dempsey, and Jonathan G Li. A unifying framework for multi-criteria fluence map optimization models. *Physics in Medicine & Biology*, 49(10):1991, 2004.

[92] David L Craft, Tarek F Halabi, Helen A Shih, and Thomas R Bortfeld. Approximating convex pareto surfaces in multiobjective radiotherapy planning. *Medical physics*, 33(9):3399–3407, 2006.

[93] Guillermo Cabrera-Guerrero, Andrew J Mason, Andrea Raith, and Matthias Ehrgott. Pareto local search algorithms for the multi-objective beam angle optimisation problem. *Journal of Heuristics*, 24(2):205–238, 2018.

[94] Mac Clements, Nicholas Schupp, Megan Tattersall, Anthony Brown, and Randy Larson. Monaco treatment planning system tools and optimization processes. *Medical Dosimetry*, 43(2):106–117, 2018.

[95] Sebastiaan Breedveld, David Craft, Rens Van Haveren, and Ben Heijmen. Multi-criteria optimization and decision-making in radiotherapy. *European Journal of Operational Research*, 277(1):1–19, 2019.

[96] Sebastiaan Breedveld, Pascal RM Storchi, and Ben JM Heijmen. The equivalence of multi-criteria methods for radiotherapy plan optimization. *Physics in Medicine & Biology*, 54(23):7199, 2009.

[97] Richard Cole and Vasilis Gkatzelis. Approximating the Nash social welfare with indivisible items. *SIAM J. Comput.*, 47(3):1211–1236, 2018.

[98] Payman Ghasemi Saghand, Hadi Charkhgard, and Changhyun Kwon. 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, 2019.

[99] Hawthorne L Beyer, Yann Dujardin, Matthew E Watts, and Hugh P Possingham. Solving conservation planning problems with integer linear programming. *Ecological Modelling*, 328:14–22, 2016.

[100] Alain Billionnet. Mathematical optimization ideas for biodiversity conservation. *European Journal of Operational Research*, 231(3):514–534, 2013.

[101] Lee Failing and Robin Gregory. Ten common mistakes in designing biodiversity indicators for forest policy. *Journal of Environmental Management*, 68(2):121 – 132, 2003.

[102] Alvaro Sierra-Altamiranda, Hadi Charkhgard, Mitchell Eaton, Julien Martin, Simeon Yurek, and Bradley J. Udell. Spatial conservation planning under uncertainty using modern portfolio theory and Nash bargaining solution. *Ecological Modelling*, 423:109016, 2020.

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

[104] John F Nash Jr. The bargaining problem. *Econometrica: Journal of the Econometric Society*, pages 155–162, 1950.

[105] Ioannis Caragiannis, David Kurokawa, Hervé Moulin, Ariel D Procaccia, Nisarg Shah, and Junxing Wang. The unreasonable fairness of maximum Nash welfare. *ACM Transactions on Economics and Computation (TEAC)*, 7(3):1–32, 2019.

[106] Vijay V Vazirani. The notion of a rational convex program, and an algorithm for the arrow-debreu Nash bargaining game. *Journal of the ACM (JACM)*, 59(2):1–36, 2012.

[107] Mostafa Abouei Ardakan, Mohammad Sima, Ali Zeinal Hamadani, and David W Coit. A novel strategy for redundant components in reliability–redundancy allocation problems. *IIE Transactions*, 48(11):1043–1057, 2016.

[108] Mohammad Feizabadi and Abdolhamid Eshraghniaye Jahromi. A new model for reliability optimization of series-parallel systems with non-homogeneous components. *Reliability Engineering & System Safety*, 157:101–112, 2017.

[109] Enze Zhang and Qingwei Chen. Multi-objective reliability redundancy allocation in an interval environment using particle swarm optimization. *Reliability Engineering & System Safety*, 145:83–92, 2016.

[110] Satya Tamby and Daniel Vanderpooten. Enumeration of the nondominated set of multiobjective discrete optimization problems. *INFORMS Journal on Computing*, 2020. Available online.

[111] Tyler Perini, Natashia Boland, Diego Pecin, and Martin Savelsbergh. A criterion space method for biobjective mixed integer programming: The boxed line method. *INFORMS Journal on Computing*, 32(1):16–39, 2020.

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

[113] Vahid Mahmoodian, Iman Dayarian, Payman Ghasemi Saghand, Yu Zhang, and Hadi Charkhgard. A criterion space branch-and-cut algorithm for mixed integer bi-linear maximum multiplicative programs. *Preprint*, 2020.

[114] Vahid Mahmoodian, Hadi Charkhgard, and Yu Zhang. Multi-objective optimization based algorithms for solving mixed integer linear minimum multiplicative programs. *Computers & Operations Research*, 128:105 – 178, 2021.

[115] Payman Ghasemi Saghand and Hadi Charkhgard. A criterion space search algorithm for mixed integer linear maximum multiplicative programs: A multi-objective optimization approach. *Preprint*, 2019.

[116] Payman Ghasemi Saghand and Hadi Charkhgard. Exact solution approaches for integer linear generalized maximum multiplicative programs through the lens of multi-objective optimization. *Preprint*, 2019.

[117] Lawrence J. Watters. Reduction of integer polynomial programming problems to zero-one linear programming problems. *Operations Research*, 15(6):1171–1174, 1967.

[118] Fred Glover and Eugene Woolsey. Technical note—converting the 0-1 polynomial programming problem to a 0-1 linear program. *Operations Research*, 22(1):180–182, 1974.

[119] Fred Glover. Improved linear integer programming formulations of nonlinear integer problems. *Management Science*, 22(4):455–460, 1975.

[120] Christopher J C Burges. Factoring as optimization. *Technical Report MSR-TR-2002-8*, August 2002. `https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-2002-83.pdf`.

[121] D Coppersmith and J Lee. Indivisibility and divisibility polytopes. In *Novel Approaches to Hard Discrete Optimization*, page 71–95. Fields Institute Communications, American Mathematical Society, Providence, Rhode Island, 2003.

[122] Jonathan Borwein and David Bailey. *Mathematics by Experiment: Plausible Reasoning in the 21st Century.* A K Peters/CRC Press, 2nd edition, 2008.

[123] Santanu S. Dey and Akshay Gupte. Analysis of MILP techniques for the pooling problem. *Operations Research*, 63(2):412–427, 2015.

[124] James Luedtke, Mahdi Namazifar, and Jeff Linderoth. Some results on the strength of relaxations of multilinear functions. *Mathematical Programming*, 136(2):325–351, 2012.

[125] Yuelin Gao, Chengxian Xu, and Yongjian Yang. An outcome-space finite algorithm for solving linear multiplicative programming. *Applied Mathematics and Computation*, 179(2):494 – 505, 2006.

[126] Hong-Seo Ryoo and Nikolaos V. Sahinidis. Global optimization of multiplicative programs. *Journal of Global Optimization*, 26(4):387–418, 2003.

[127] Pavel Stavrev, D Hristov, B Warkentin, E Sham, N Stavreva, and BG Fallone. Inverse treatment planning by physically constrained minimization of a biological objective function. *Medical physics*, 30(11):2948–2958, 2003.

[128] M Alber and F Nüsslin. A representation of an NTCP function for local complication mechanisms. *Physics in Medicine & Biology*, 46(2):439, 2001.

[129] M Zaider and GN Minerbo. Tumour control probability: a formulation applicable to any temporal protocol of dose delivery. *Physics in Medicine & Biology*, 45(2):279, 2000.

[130] J. Löf, Stockholms universitet. Department of Medical Radiation Physics, and Cancerföreningen i Stockholm. *Development of a General Framework for Optimization of Radiation Therapy*. Department of Medical Radiation Physics, 2000.

[131] Taewoo Lee, Muhannad Hammad, Timothy C. Y. Chan, Tim Craig, and Michael B. Sharpe. Predicting objective function weights from patient anatomy in prostate IMRT treatment planning. *Medical Physics*, 40(12):121706, 2013.

[132] Justin J. Boutilier, Taewoo Lee, Tim Craig, Michael B. Sharpe, and Timothy C. Y. Chan. Models for predicting objective function weights in prostate cancer IMRT. *Medical Physics*, 42(4):1586–1595, 2015.

[133] Saeed Ahmed, Benjamin Nelms, Dawn Gintz, Jimmy Caudell, Geoffrey Zhang, Eduardo G Moros, and Vladimir Feygelman. A method for a priori estimation of best feasible DVH for organs-at-risk: Validation for head and neck VMAT planning. *Medical physics*, 44(10):5486–5497, 2017.

[134] Andreas Darmann and Joachim Schauer. Maximizing nash product social welfare in allocating indivisible goods. *European Journal of Operational Research*, 247(2):548 – 559, 2015.

[135] Trung Thanh Nguyen and Jörg Rothe. Minimizing envy and maximizing average nash social welfare in the allocation of indivisible goods. *Discrete Applied Mathematics*, 179:54 – 68, 2014.

[136] Richard Cole and Vasilis Gkatzelis. Approximating the nash social welfare with indivisible items. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 371–380, New York, NY, USA, 2015. ACM.

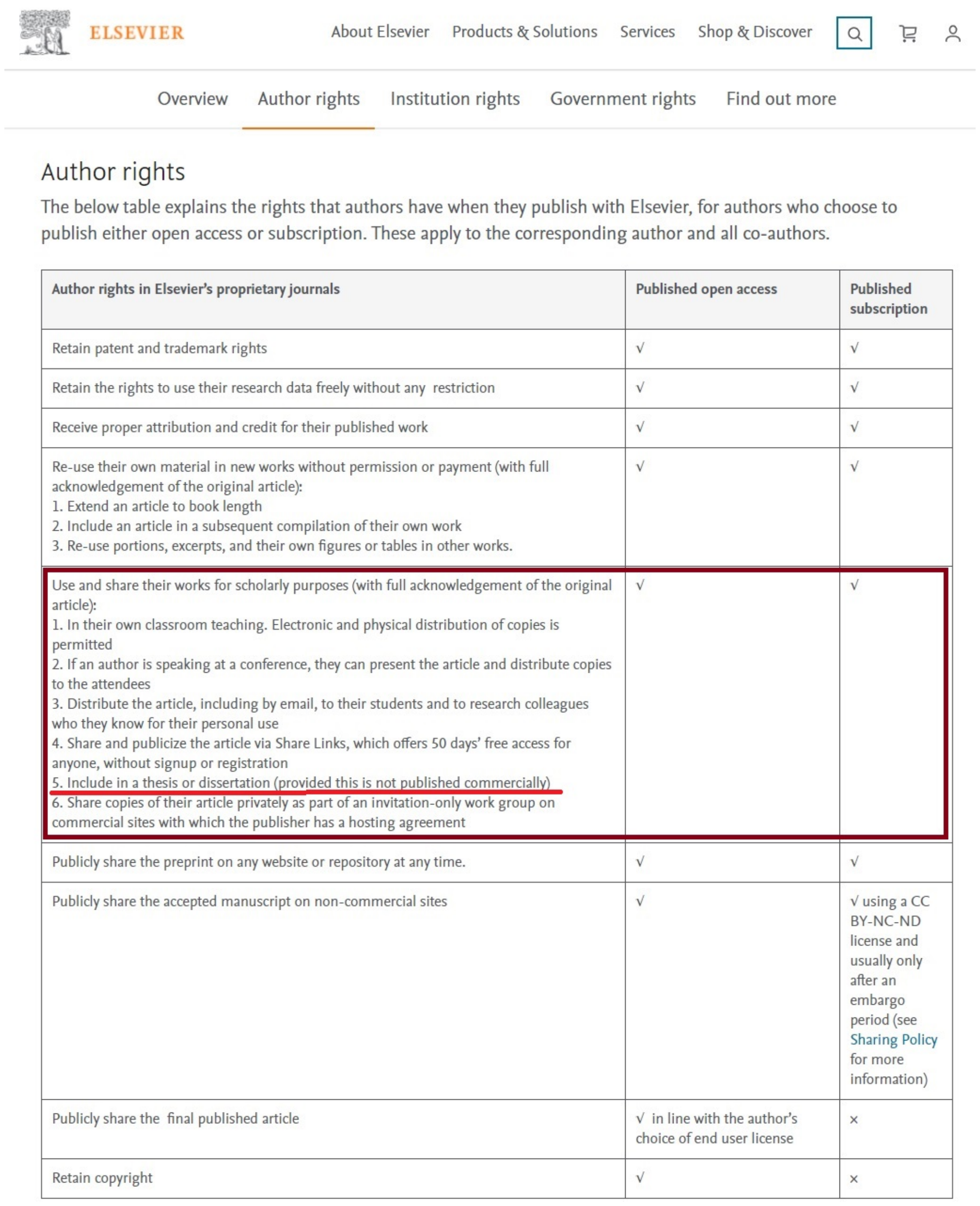
[137] Brandon Fain, Kamesh Munagala, and Nisarg Shah. Fair allocation of indivisible public goods. In *Proceedings of the 2018 ACM Conference on Economics and Computation*, EC '18, pages 575–592, New York, NY, USA, 2018. ACM.

[138] Sara Retal and Abdellah Idrissi. A multi-objective optimization system for mobile gateways selection in vehicular ad-hoc networks. *Computers & Electrical Engineering*, 73:289 – 303, 2019.

[139] Jennifer Mendoza-Alonzo, JosÃ© Zayas-Castro, and Hadi Charkhgard. Office-based and home-care for older adults in primary care: A comparative analysis using the nash bargaining solution. *Socio-Economic Planning Sciences*, page 100710, 2019.

Appendix A: Copyright Permissions

A1: Reprint Permission for Chapter 2



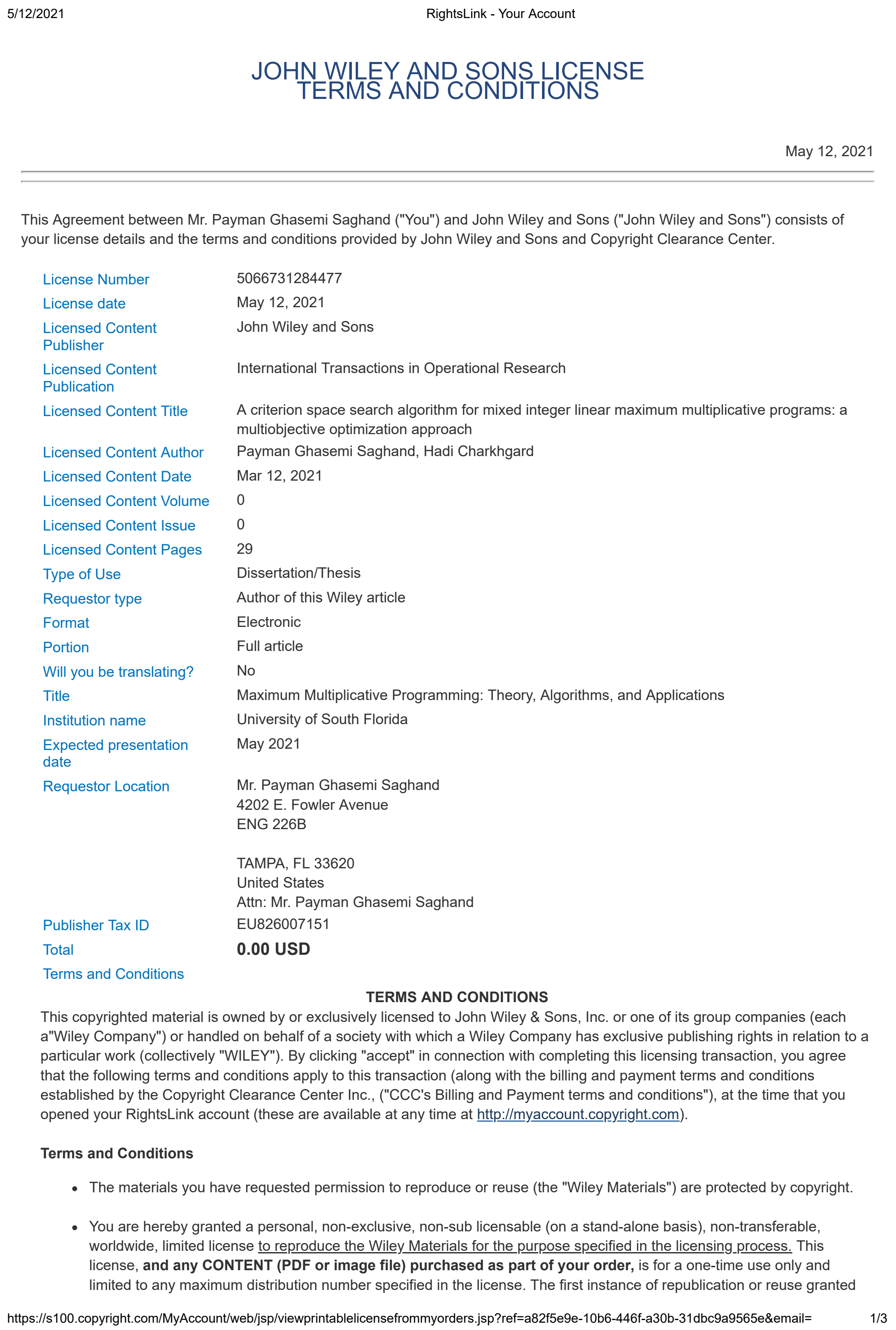
Author rights

The below table explains the rights that authors have when they publish with Elsevier, for authors who choose to publish either open access or subscription. These apply to the corresponding author and all co-authors.

Author rights in Elsevier's proprietary journals	Published open access	Published subscription
Retain patent and trademark rights	√	√
Retain the rights to use their research data freely without any restriction	√	√
Receive proper attribution and credit for their published work	√	√
Re-use their own material in new works without permission or payment (with full acknowledgement of the original article): 1. Extend an article to book length 2. Include an article in a subsequent compilation of their own work 3. Re-use portions, excerpts, and their own figures or tables in other works.	√	√
Use and share their works for scholarly purposes (with full acknowledgement of the original article): 1. In their own classroom teaching. Electronic and physical distribution of copies is permitted 2. If an author is speaking at a conference, they can present the article and distribute copies to the attendees 3. Distribute the article, including by email, to their students and to research colleagues who they know for their personal use 4. Share and publicize the article via Share Links, which offers 50 days' free access for anyone, without signup or registration 5. Include in a thesis or dissertation (provided this is not published commercially) 6. Share copies of their article privately as part of an invitation-only work group on commercial sites with which the publisher has a hosting agreement	√	√
Publicly share the preprint on any website or repository at any time.	√	√
Publicly share the accepted manuscript on non-commercial sites	√	√ using a CC BY-NC-ND license and usually only after an embargo period (see Sharing Policy for more information)
Publicly share the final published article	√ in line with the author's choice of end user license	×
Retain copyright	√	×

https://www.elsevier.com/about/policies/copyright#Author-rights

A2: Reprint Permission for Chapter 3



5/12/2021 RightsLink - Your Account

JOHN WILEY AND SONS LICENSE TERMS AND CONDITIONS

May 12, 2021

This Agreement between Mr. Payman Ghasemi Saghand ("You") and John Wiley and Sons ("John Wiley and Sons") consists of your license details and the terms and conditions provided by John Wiley and Sons and Copyright Clearance Center.

License Number	5066731284477
License date	May 12, 2021
Licensed Content Publisher	John Wiley and Sons
Licensed Content Publication	International Transactions in Operational Research
Licensed Content Title	A criterion space search algorithm for mixed integer linear maximum multiplicative programs: a multiobjective optimization approach
Licensed Content Author	Payman Ghasemi Saghand, Hadi Charkhgard
Licensed Content Date	Mar 12, 2021
Licensed Content Volume	0
Licensed Content Issue	0
Licensed Content Pages	29
Type of Use	Dissertation/Thesis
Requestor type	Author of this Wiley article
Format	Electronic
Portion	Full article
Will you be translating?	No
Title	Maximum Multiplicative Programming: Theory, Algorithms, and Applications
Institution name	University of South Florida
Expected presentation date	May 2021
Requestor Location	Mr. Payman Ghasemi Saghand 4202 E. Fowler Avenue ENG 226B TAMPA, FL 33620 United States Attn: Mr. Payman Ghasemi Saghand
Publisher Tax ID	EU826007151
Total	**0.00 USD**
Terms and Conditions	

TERMS AND CONDITIONS

This copyrighted material is owned by or exclusively licensed to John Wiley & Sons, Inc. or one of its group companies (each a"Wiley Company") or handled on behalf of a society with which a Wiley Company has exclusive publishing rights in relation to a particular work (collectively "WILEY"). By clicking "accept" in connection with completing this licensing transaction, you agree that the following terms and conditions apply to this transaction (along with the billing and payment terms and conditions established by the Copyright Clearance Center Inc., ("CCC's Billing and Payment terms and conditions"), at the time that you opened your RightsLink account (these are available at any time at http://myaccount.copyright.com).

Terms and Conditions

- The materials you have requested permission to reproduce or reuse (the "Wiley Materials") are protected by copyright.

- You are hereby granted a personal, non-exclusive, non-sub licensable (on a stand-alone basis), non-transferable, worldwide, limited license to reproduce the Wiley Materials for the purpose specified in the licensing process. This license, **and any CONTENT (PDF or image file) purchased as part of your order,** is for a one-time use only and limited to any maximum distribution number specified in the license. The first instance of republication or reuse granted

https://s100.copyright.com/MyAccount/web/jsp/viewprintablelicensefrommyorders.jsp?ref=a82f5e9e-10b6-446f-a30b-31dbc9a9565e&email= 1/3 by this license must be completed within two years of the date of the grant of this license (although copies prepared before the end date may be distributed thereafter). The Wiley Materials shall not be used in any other manner or for any other purpose, beyond what is granted in the license. Permission is granted subject to an appropriate acknowledgement given to the author, title of the material/book/journal and the publisher. You shall also duplicate the copyright notice that appears in the Wiley publication in your use of the Wiley Material. Permission is also granted on the understanding that nowhere in the text is a previously published source acknowledged for all or part of this Wiley Material. Any third party content is expressly excluded from this permission.

- With respect to the Wiley Materials, all rights are reserved. Except as expressly granted by the terms of the license, no part of the Wiley Materials may be copied, modified, adapted (except for minor reformatting required by the new Publication), translated, reproduced, transferred or distributed, in any form or by any means, and no derivative works may be made based on the Wiley Materials without the prior permission of the respective copyright owner.**For STM Signatory Publishers clearing permission under the terms of the STM Permissions Guidelines only, the terms of the license are extended to include subsequent editions and for editions in other languages, provided such editions are for the work as a whole in situ and does not involve the separate exploitation of the permitted figures or extracts,** You may not alter, remove or suppress in any manner any copyright, trademark or other notices displayed by the Wiley Materials. You may not license, rent, sell, loan, lease, pledge, offer as security, transfer or assign the Wiley Materials on a stand-alone basis, or any of the rights granted to you hereunder to any other person.

- The Wiley Materials and all of the intellectual property rights therein shall at all times remain the exclusive property of John Wiley & Sons Inc, the Wiley Companies, or their respective licensors, and your interest therein is only that of having possession of and the right to reproduce the Wiley Materials pursuant to Section 2 herein during the continuance of this Agreement. You agree that you own no right, title or interest in or to the Wiley Materials or any of the intellectual property rights therein. You shall have no rights hereunder other than the license as provided for above in Section 2. No right, license or interest to any trademark, trade name, service mark or other branding ("Marks") of WILEY or its licensors is granted hereunder, and you agree that you shall not assert any such right, license or interest with respect thereto

- NEITHER WILEY NOR ITS LICENSORS MAKES ANY WARRANTY OR REPRESENTATION OF ANY KIND TO YOU OR ANY THIRD PARTY, EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO THE MATERIALS OR THE ACCURACY OF ANY INFORMATION CONTAINED IN THE MATERIALS, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTY OF MERCHANTABILITY, ACCURACY, SATISFACTORY QUALITY, FITNESS FOR A PARTICULAR PURPOSE, USABILITY, INTEGRATION OR NON-INFRINGEMENT AND ALL SUCH WARRANTIES ARE HEREBY EXCLUDED BY WILEY AND ITS LICENSORS AND WAIVED BY YOU.

- WILEY shall have the right to terminate this Agreement immediately upon breach of this Agreement by you.

- You shall indemnify, defend and hold harmless WILEY, its Licensors and their respective directors, officers, agents and employees, from and against any actual or threatened claims, demands, causes of action or proceedings arising from any breach of this Agreement by you.

- IN NO EVENT SHALL WILEY OR ITS LICENSORS BE LIABLE TO YOU OR ANY OTHER PARTY OR ANY OTHER PERSON OR ENTITY FOR ANY SPECIAL, CONSEQUENTIAL, INCIDENTAL, INDIRECT, EXEMPLARY OR PUNITIVE DAMAGES, HOWEVER CAUSED, ARISING OUT OF OR IN CONNECTION WITH THE DOWNLOADING, PROVISIONING, VIEWING OR USE OF THE MATERIALS REGARDLESS OF THE FORM OF ACTION, WHETHER FOR BREACH OF CONTRACT, BREACH OF WARRANTY, TORT, NEGLIGENCE, INFRINGEMENT OR OTHERWISE (INCLUDING, WITHOUT LIMITATION, DAMAGES BASED ON LOSS OF PROFITS, DATA, FILES, USE, BUSINESS OPPORTUNITY OR CLAIMS OF THIRD PARTIES), AND WHETHER OR NOT THE PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION SHALL APPLY NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY PROVIDED HEREIN.

- Should any provision of this Agreement be held by a court of competent jurisdiction to be illegal, invalid, or unenforceable, that provision shall be deemed amended to achieve as nearly as possible the same economic effect as the original provision, and the legality, validity and enforceability of the remaining provisions of this Agreement shall not be affected or impaired thereby.

- The failure of either party to enforce any term or condition of this Agreement shall not constitute a waiver of either party's right to enforce each and every term and condition of this Agreement. No breach under this agreement shall be deemed waived or excused by either party unless such waiver or consent is in writing signed by the party granting such waiver or consent. The waiver by or consent of a party to a breach of any provision of this Agreement shall not operate or be construed as a waiver of or consent to any other or subsequent breach by such other party.

- This Agreement may not be assigned (including by operation of law or otherwise) by you without WILEY's prior written consent.

- Any fee required for this permission shall be non-refundable after thirty (30) days from receipt by the CCC.

- These terms and conditions together with CCC's Billing and Payment terms and conditions (which are incorporated herein) form the entire agreement between you and WILEY concerning this licensing transaction and (in the absence of fraud) supersedes all prior agreements and representations of the parties, oral or written. This Agreement may not be amended except in writing signed by both parties. This Agreement shall be binding upon and inure to the benefit of the parties' successors, legal representatives, and authorized assigns.

- In the event of any conflict between your obligations established by these terms and conditions and those established by CCC's Billing and Payment terms and conditions, these terms and conditions shall prevail.

- WILEY expressly reserves all rights not specifically granted in the combination of (i) the license details provided by you and accepted in the course of this licensing transaction, (ii) these terms and conditions and (iii) CCC's Billing and Payment terms and conditions.

- This Agreement will be void if the Type of Use, Format, Circulation, or Requestor Type was misrepresented during the licensing process.

- This Agreement shall be governed by and construed in accordance with the laws of the State of New York, USA, without regards to such state's conflict of law rules. Any legal action, suit or proceeding arising out of or relating to these Terms and Conditions or the breach thereof shall be instituted in a court of competent jurisdiction in New York County in the State of New York in the United States of America and each party hereby consents and submits to the personal jurisdiction of such court, waives any objection to venue in such court and consents to service of process by registered or certified mail, return receipt requested, at the last known address of such party.

WILEY OPEN ACCESS TERMS AND CONDITIONS
Wiley Publishes Open Access Articles in fully Open Access Journals and in Subscription journals offering Online Open. Although most of the fully Open Access journals publish open access articles under the terms of the Creative Commons Attribution (CC BY) License only, the subscription journals and a few of the Open Access Journals offer a choice of Creative Commons Licenses. The license type is clearly identified on the article.
The Creative Commons Attribution License
The Creative Commons Attribution License (CC-BY) allows users to copy, distribute and transmit an article, adapt the article and make commercial use of the article. The CC-BY license permits commercial and non-
Creative Commons Attribution Non-Commercial License
The Creative Commons Attribution Non-Commercial (CC-BY-NC)License permits use, distribution and reproduction in any medium, provided the original work is properly cited and is not used for commercial purposes.(see below)

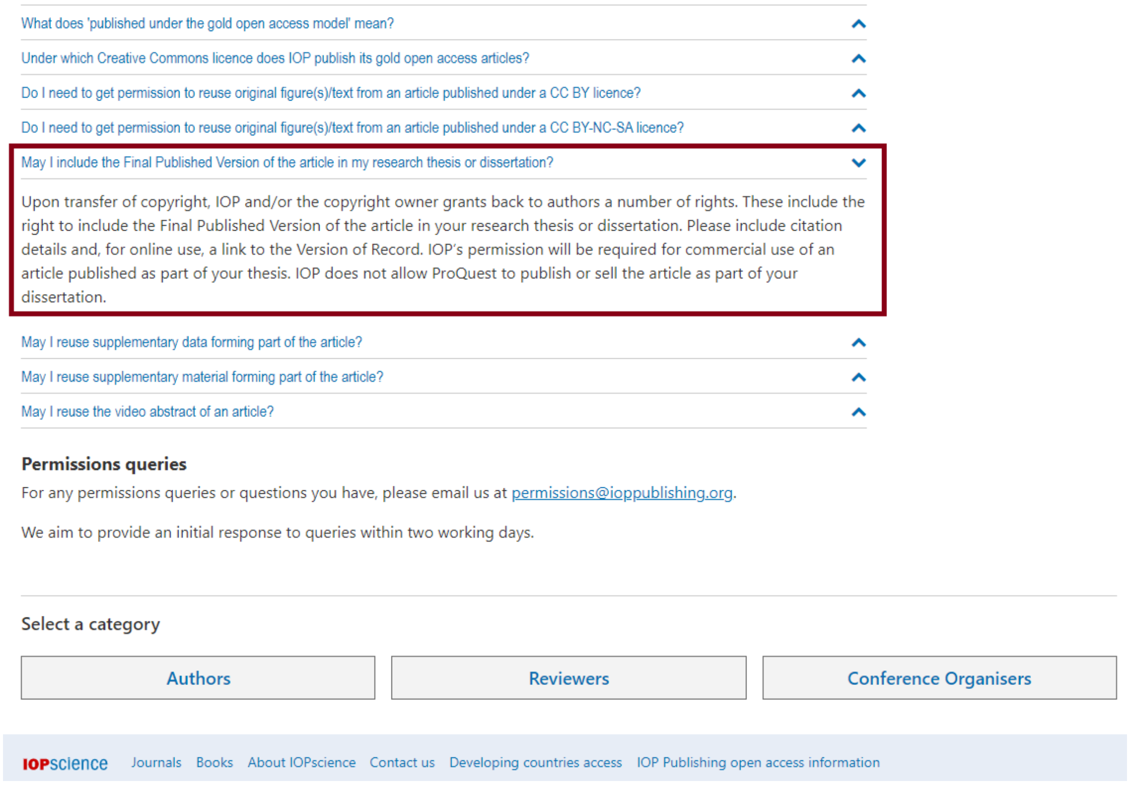
Creative Commons Attribution-Non-Commercial-NoDerivs License
The Creative Commons Attribution Non-Commercial-NoDerivs License (CC-BY-NC-ND) permits use, distribution and reproduction in any medium, provided the original work is properly cited, is not used for commercial purposes and no modifications or adaptations are made. (see below)
Use by commercial "for-profit" organizations
Use of Wiley Open Access articles for commercial, promotional, or marketing purposes requires further explicit permission from Wiley and will be subject to a fee.
Further details can be found on Wiley Online Library http://olabout.wiley.com/WileyCDA/Section/id-410895.html

Other Terms and Conditions:

v1.10 Last updated September 2015

Questions? customercare@copyright.com or +1-855-239-3415 (toll free in the US) or +1-978-646-2777.

A3: Reprint Permission for Chapter 5



What does 'published under the gold open access model' mean?

Under which Creative Commons licence does IOP publish its gold open access articles?

Do I need to get permission to reuse original figure(s)/text from an article published under a CC BY licence?

Do I need to get permission to reuse original figure(s)/text from an article published under a CC BY-NC-SA licence?

May I include the Final Published Version of the article in my research thesis or dissertation?

Upon transfer of copyright, IOP and/or the copyright owner grants back to authors a number of rights. These include the right to include the Final Published Version of the article in your research thesis or dissertation. Please include citation details and, for online use, a link to the Version of Record. IOP's permission will be required for commercial use of an article published as part of your thesis. IOP does not allow ProQuest to publish or sell the article as part of your dissertation.

May I reuse supplementary data forming part of the article?

May I reuse supplementary material forming part of the article?

May I reuse the video abstract of an article?

Permissions queries

For any permissions queries or questions you have, please email us at permissions@ioppublishing.org.

We aim to provide an initial response to queries within two working days.

Select a category

https://publishingsupport.iopscience.iop.org/permissions-faqs/

Appendix B: Chapter 3

B1: Detailed Comparison Between B&B and Algorithm 3

Table B1.1: Performance comparison between Algorithm 3 and B&B on pure binary and mixed binary instances with $p = 2$

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

Table B1.2: Performance comparison between Algorithm 3 and B&B on pure integer and mixed integer instances with $p = 2$

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

B2: Detailed Comparison Between Algorithm 3 and SOCP for $p = 3$

Table B2.1: Performance comparison between Algorithm 3 and SOCP on pure binary and mixed binary instances with $p = 3$

	Pure binary					Mixed binary			
	Algorithm 3		SOCP			Algorithm 3		SOCP	
$m \times n$	**#N**	**T(sec.)**	**#S**	**%G**	**T(sec.)**	**#N**	**T(sec.)**	**%G**	**T(sec.)**
200 × 100	19.20	2.57	7	0	10.13	11.70	1.34	0	2.34
200 × 200	17.10	6.82	7	0	38.29	12	2.39	0	5.53
200 × 300	15.60	13.24	7	0	87.59	12.60	3.77	0	10.58
200 × 400	18	28.84	8	0	219.55	12.30	5.10	0	14.76
Avg	17.48	12.87	7.25	0	93.39	12.15	3.15	0	8.30
400 × 200	16.40	16.48	2	0	53.66	11.40	6.44	0	14.84
400 × 400	13.80	43.54	2	0	93.20	11.40	11.98	0	37.98
400 × 600	16.50	177.93	1	0	330.82	11.10	18.52	0	62.40
400 × 800	15.30	129.70	4	0	967.47	10.20	23.47	0	90.17
Avg	15.50	91.92	2.25	0	499.38	11.03	15.10	0	51.35
600 × 300	21.30	64.60	0	-	-	11.40	15.79	0	43.69
600 × 600	14.10	341.89	0	-	-	10.50	29.86	0	132.32
600 × 900	12.50	1,829.57	0	-	-	10.50	49.83	0	212.06
600 × 1200	14.10	1,611.47	2	61	7,200	10.50	65.24	0	359.56
Avg	15.50	961.88	0.50	61	7,200	10.73	40.18	0	186.91
800 × 400	15	61.13	2	0	538.43	11.70	37.72	0	135.12
800 × 800	15	571.09	0	-	-	12	75.93	0	284.44
800 × 1200	10.60	1,534.63	5	69	7,200	10.50	111.03	0	485.98
800 × 1600	12.60	904.61	7	78	7,200	9.90	147.69	0	917.01
Avg	13.30	767.87	3.50	63.64	6,248.34	11.03	93.09	0	455.63

Table B2.2: Performance comparison between Algorithm 3 and SOCP on pure integer and mixed integer instances with $p = 3$

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

Table B2.3: Performance comparison between Algorithm 3 and SOCP on instances with $p = 3$ for the convergence tolerance of 10^{-6}

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

B3: Detailed Comparison Between Algorithm 3 and SOCP for $p = 4$

Table B3.1: Performance comparison between Algorithm 3 and SOCP on pure binary and mixed binary instances with $p = 4$

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

Table B3.2: Performance comparison between Algorithm 3 and SOCP on pure integer and mixed integer instances with $p = 4$

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

Table B3.3: Performance comparison between Algorithm 3 and SOCP on instances with $p = 4$ for the convergence tolerance of 10^{-6}

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

Appendix C: Chapter 4

C1: A Detailed Comparison Between SOCP and CFSSA-I on Instances with Unit Geometric Weights

In Tables C1.1 and C1.2, '#IP' is the average number of single-objective integer linear programs solved by CFSSA-I, and 'T(sec.)' is the average solution time in seconds.

Table C1.1: Pure binary instances with unit geometric weights

	$p = 2$			$p = 3$			$p = 4$		
$m \times n$	**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 C1.2: 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

C2: Case Study

In this section, we show the performance of our proposed solution methods on three different problems arising in the fields of game theory, transportation, and health-care.

C2.1: Fair Allocation of Indivisible Goods

The allocation of indivisible goods is the problem of allocating a set of indivisible items, such as artwork or jewelry, among a set of (independent) agents. In this problem, every agent has a utility function that they would like to maximize during the allocation process. The main issue arising in this context is to guarantee the fairness of the allocation. [12] showed that the highest provable fairness is achievable by maximizing the Nash Social Welfare, which is the multiplication of all utility functions. This combined with the fact that agents are mostly assumed to have additive utilities across the goods in the relevant literature (see for instance [12, 134, 135, 136, 137]) gives rise to IL-GMMP with unit geometric weights.

Specifically, binary decision variables are required for identifying whether item i should be assigned to person j or not. Also, some linear constraints are needed for ensuring that each item will be assigned to exactly one person. Every agent is assumed to have a linear utility function of goods where the coefficients in the function represent the preference that each agent has for each good. We note that the linearity of each utility function is the result of the additivity of the utilities. Moreover, in order to guarantee the fairness, the utility functions are assumed to be strictly positive, i.e. every agent should receive at least one item.

As for our case study, we consider the problem of allocating 50 goods to 5 agents, which results in an IL-GMMP with $p = 5$, 250 binary variables, and 50 equality constraints. Every agent is required to choose a preference of $\{1, 2, 3\}$ for each item. In order to eliminate the need for the normalization of the utility functions, the summation of the preferences of each agent over the 50 goods should be twice the number of goods. For the instance, we tested our best approach CFSSA-I (when employing CPLEX) against the SOCP solver of CPLEX, Gurobi, and Xpress by imposing a time limit of 3,600 seconds and using a single thread. The solution times and relative optimality gaps are presented in Table C2.1. In this table, columns labeled 'T (sec.)' show the solution time of a method, and columns labeled '%Gap' show relative optimality gap obtained by a method.

Table C2.1: Performance comparison for the fair allocation of indivisible goods problem

	CFSSA-I		**SOCP**					
	CPLEX		CPLEX		Gurobi		Xpress	
$\boldsymbol{\pi}$	T(sec.)	%Gap	T(sec.)	%Gap	T(sec.)	%Gap	T(sec.)	%Gap
$(1, 1, 1, 1, 1)$	21.06	0	3,600	0.01	–	–	3,600	0.01

Observe that our proposed approach is the best choice. In fact, the SOCP solver of CPLEX and Xpress were not able to close the optimality gap within the imposed time limit. Also, we did not report any result for SOCP of Gurobi because it shortly faced some issues and returned a solution that its objective value was very far from the optimal one.

C2.2: Mobile Gateway Selection in Intelligent Transportation Systems

In the field of Intelligent Transportation Systems, permanent access of Client Vehicles (CVs) to the Internet is crucial. However, due to the speed of CVs, it would be difficult to create a stable Internet access through fixed/stationary gateways. This is because CVs go in and out of the communication range of a gateway quickly. A solution proposed in the literature for resolving this issue is to employ Mobile Gateways (MGs). Ideally, we would like to assign all CVs to all MGs but that is typically impossible since it results in overload issues for some MGs. So, a key problem in Intelligent Transportation Systems is to identify the best utilization of MGs.

To solve this problem, [138] proposed a bi-objective binary linear programming model where the goals are maximizing the number of CVs that are connected to MGs and minimizing the amount of traffic handled by the selected MGs. Their model includes several linear constraints which assure that each CV is connected to at most one MG. The constraints further assure that, if a CV is connected to a MG, they are within a predefined distance of each other, the difference between their velocities does not surpass a threshold, and they are moving in the same direction. [138] proposed to solve the bi-objective model by transforming it into an IL-GMMP (with $p = 2$). They showed the effectiveness of their method over a variety of instances for different preference/geometric weights including $\boldsymbol{\pi} = (1, 1)$, $\boldsymbol{\pi} = (0.3, 0.7)$, and $\boldsymbol{\pi} = (0.7, 0.3)$.

The authors of [138] kindly provided their data to us. Hence, for this case study, we were able to generate one of their largest instances, which includes 5775 binary variables and 313775 constraints. However, by employing a simple variable-fixing (or preprocessing) technique, the size of their model can be reduced to 2540 binary variables and 22111 constraints. Again, we tested our best approach CFSSA-I (when employing CPLEX) against the SOCP solver of CPLEX, Gurobi, and Xpress by imposing a time limit of 3,600 seconds and using a single thread. The solution times and relative optimality gaps are presented in Table C2.2. It is obvious that our approach has been able to solve the instance for different geometric weights very quickly to optimality. The second best approach is the SOCP solver of Xpress which is significantly slower than our proposed approach.

Table C2.2: Performance comparison of different solution approaches on the mobile gateway selection problem

	CFSSA-I		**SOCP**					
	CPLEX		CPLEX		Gurobi		Xpress	
$\boldsymbol{\pi}$	T(sec.)	%Gap	T(sec.)	%Gap	T(sec.)	%Gap	T(sec.)	%Gap
$(1\ ,\ 1)$	4.44	0	3,600	18	32.08	0	202.09	0
$(0.3, 0.7)$	4.58	0	3,600	2	3600	–	2060.6	0
$(0.7, 0.3)$	101.57	0	3,600	25	3600	–	3600	2×10^{-3}

C2.3: Primary Care Selection Problem for Older Adults

The population of people with the age of 65 or more is increasing in the world. Therefore, the World Health Organization (WHO) is emphasizing the primary health care for dealing with the aging population. In the United States (US), there are three main primary care delivery settings including home-based care, office-based care, and mixed-care. For a given geographical region and primary care practice, identifying which delivery setting is the best strategy is a key problem. This problem is challenging since a health care system has multiple players including the organization, patients, and providers. These players have conflicting objectives. For example, an organization is typically interested in reducing its total cost, patients would like to improve their experience, e.g. not to be rejected and/or be assigned to an unpreferred care delivery setting, providers (such as physicians and nurses) would also like to improve their experience, e.g. not to have a high utilization rate.

To address this problem, [139] proposed a four objective optimization model that attempts to identify which patients should be served, when they should be served, through which primary care setting they should be served, and by whom they should be served. Their optimization model is a pure integer linear program. To solve the problem, they proposed to use the Nash bargaining solution. Therefore, they re-formulated the problem as a IL-GMMP with unit geometric weights.

The authors of [139] kindly provided their data to use. Hence, for this case study, we were able to generate one of their large instances, which includes 4100 binary and 484 integer variables, and 4834 constraints. Note that, in order to use our proposed algorithms, we converted all integer variables to binaries, resulting in a problem with a total of 5520 binary variables and 5318 constraints. Similar to the other two case studies, we solved the instance using CFSSA-I (when employing CPLEX) and compared it against the SOCP solver of CPLEX, Gurobi, and Xpress. We imposed a time limit of 3,600 seconds and used a single thread. The solution times and relative optimality gaps are presented in Table C2.3. Again, we observe that, our proposed method has preformed significantly better than any other method.

Table C2.3: Performance comparison for the primary care selection problem

	CFSSA-I		**SOCP**					
	CPLEX		CPLEX		Gurobi		Xpress	
$\boldsymbol{\pi}$	T(sec.)	%Gap	T(sec.)	%Gap	T(sec.)	%Gap	T(sec.)	%Gap
$(1,1,1,1)$	7.8	0	3600	14	3600	0.08	453	0

Appendix D: Chapter 5

D1: Implementing Lexico and 2pϵc on the Liver Case

In this section, we provide the details of how we have specifically implemented Lexico and 2pϵ on the liver case, i.e., Section 5.5.2. Before doing so, we provide some notations to facilitate the presentation. First note that based on the wish-list given in Table 5.3, three penalty objectives should be created. The first one belongs to the PTV (which has the highest priority), the second one belongs to the liver (which has the second highest priority), and the last one belongs to all other organs together (which means that none of the other organs has priority over each other). With this in mind, we denote the objective with priority $i \in \{1,2,3\}$ by $\bar{f}_i(\boldsymbol{d})$ and define them as follows,

$$\begin{aligned}
\bar{f}_1(\boldsymbol{d}) &:= \sum_{v \in V_{PTV}} (d_v - 55)^2, \\
\bar{f}_2(\boldsymbol{d}) &:= \sum_{v \in V_{Liver}} (d_v - 0)^2, \\
\bar{f}_3(\boldsymbol{d}) &:= \sum_{\substack{s \in S/ \\ \{PTV, Liver\}}} \sum_{v \in V_s} (d_v - 0)^2.
\end{aligned}$$

We denote the goal values given in Table 5.3 by g_i for each $i \in \{1,2,3\}$. Specifically, we have that,

$$\begin{aligned}
g_1 &= 68,386.38, \\
g_2 &= 8,750,124.63, \\
g_3 &= 42,012,837.22.
\end{aligned}$$

Finally, we define δ as a constant slightly greater than one to avoid numerical issues in Lexico and 2pϵc. In our implementation, we set $\delta = 1.03$ as suggested by [96].

D1.1: Lexico

The Lexico contains three main steps and in each step, one optimization problem needs to be solved. The first step optimizes the penalty function with the highest priority over all feasible solutions, i.e.,

$$\bar{f}_1^* := \min\{f_1(\boldsymbol{d}) : \boldsymbol{d} \in \mathcal{D}\}.$$

In the second step, the second penalty function is optimized over all feasible solutions that are optimal for the first objective,

$$\bar{f}_2^* := \min\{f_2(\boldsymbol{d}) : f_1(\boldsymbol{d}) \leqslant \bar{f}_1^*\delta \ , \ \boldsymbol{d} \in \mathcal{D}\}.$$

In the third step, the third penalty function is optimized over all feasible solutions that are optimal for the second objective among all optimal solutions for the first objective,

$$\boldsymbol{d}^*_{\text{Lexico}} \in \arg\min\{f_3(\boldsymbol{d}) : f_1(\boldsymbol{d}) \leqslant \bar{f}_1^*\delta \ , \ f_2(\boldsymbol{d}) \leqslant \bar{f}_2^*\delta \ , \ \boldsymbol{d} \in \mathcal{D}\}.$$

The solution $\boldsymbol{d}^*_{\text{Lexico}}$ is the final outcome of the Lexico which we used to make Figures 5.7 and 5.8.

D1.2: 2pϵc

The 2pϵc approach is developed by [96] and is a combination of Lexico and goal programming. The underling idea of this approach comes from this observation that Leixco tends to generate extreme solutions mainly because in each step it always searches among optimal solutions of the previous step. However, in practice, that is only necessary if the optimal objective value of the previous step is larger (i.e., worse) than the goal defined for that step. Otherwise, searching among solutions satisfying the goal can be sufficient in practice. With this in mind, 2pϵc, as the name suggests, consists of two phases. The first phase is similar to the Lexico with the only difference being in the right-hand-side values of the constraints imposed in each step. Specifically, the following three optimization problems need to be solved:

$$\begin{aligned}
&\bar{f}_1^* := \min\{f_1(\boldsymbol{d}) : \boldsymbol{d} \in \mathcal{D}\},\\
&\bar{f}_2^* := \min\{f_2(\boldsymbol{d}) : f_1(\boldsymbol{d}) \leqslant \max\{\bar{f}_1^*\delta, g_1\}\ ,\ \boldsymbol{d} \in \mathcal{D}\},\\
&\bar{f}_3^* := \min\{f_3(\boldsymbol{d}) : f_1(\boldsymbol{d}) \leqslant \max\{\bar{f}_1^*\delta, g_1\}\ ,\ f_2(\boldsymbol{d}) \leqslant \max\{\bar{f}_2^*\delta, g_2\}\ ,\ \boldsymbol{d} \in \mathcal{D}\}.
\end{aligned}$$

The issue about the first phase is that the solution corresponding to $\bar{f}_3^*$ may not be Pareto-optimal if $\max\{\bar{f}_1^*\delta, g_1\} \neq \bar{f}_1^*\delta$ or $\max\{\bar{f}_2^*\delta, g_2\} \neq \bar{f}_2^*\delta$. So, the second step is designed to ensure that a Pareto-optimal solution will be generated. Similar to the first phase, the second phase will also have three steps and in each step one optimization problem needs to be solved. The optimization problems are as follows:

$$\begin{aligned}
&\bar{\bar{f}}_1^* := \min\{f_1(\boldsymbol{d}) : f_2(\boldsymbol{d}) \leqslant \max\{\bar{f}_2^*\delta, g_2\}\ ,\ f_3(\boldsymbol{d}) \leqslant \max\{\bar{f}_3^*\delta, g_3\}\ ,\ \boldsymbol{d} \in \mathcal{D}\},\\
&\bar{\bar{f}}_2^* := \min\{f_2(\boldsymbol{d}) : f_1(\boldsymbol{d}) \leqslant \bar{\bar{f}}_1^*\delta\ ,\ f_3(\boldsymbol{d}) \leqslant \max\{\bar{f}_3^*\delta, g_3\}\ ,\ \boldsymbol{d} \in \mathcal{D}\},\\
&\boldsymbol{d}_{2\text{p}\epsilon\text{c}}^* \in \arg\min\{f_3(\boldsymbol{d}) : f_1(\boldsymbol{d}) \leqslant \bar{\bar{f}}_1^*\delta\ ,\ f_2(\boldsymbol{d}) \leqslant \bar{\bar{f}}_2^*\delta\ ,\ \boldsymbol{d} \in \mathcal{D}\}.
\end{aligned}$$

The solution $\boldsymbol{d}_{2\text{p}\epsilon\text{c}}^*$ is the final outcome of 2pϵc which we used it to make Figures 5.7 and 5.8. We note that some steps of the second phase may be redundant and therefore can be skipped. For example, if $\max\{\bar{f}_1^*\delta, g_1\} = \bar{f}_1^*\delta$ and $\max\{\bar{f}_2^*\delta, g_2\} = \bar{f}_2^*\delta$ then the solution corresponding to $\bar{f}_3^*$ is Pareto-optimal and can be named as $\boldsymbol{d}_{2\text{p}\epsilon\text{c}}^*$. So, in that case there is no need to call the second phase at all. If $\max\{\bar{f}_2^*\delta, g_2\} = \bar{f}_2^*\delta$ and $\max\{\bar{f}_3^*\delta, g_3\} = \bar{f}_3^*\delta$ then the solution corresponding to $\bar{\bar{f}}_1^*$ is Pareto-optimal and can be named as $\boldsymbol{d}_{2\text{p}\epsilon\text{c}}^*$. So, in that case there is no need to call steps 2 and 3 of the second phase at all. Finally, if $\max\{\bar{f}_3^*\delta, g_3\} = \bar{f}_3^*\delta$ then the solution corresponding to $\bar{\bar{f}}_2^*$ is Pareto-optimal and can be named as $\boldsymbol{d}^*_{2\mathrm{p}\epsilon\mathrm{c}}$. So, in that case there is no need to call step 3 of the second phase at all.

Appendix E: Chapter 6

E1: Proofs

Proof of Proposition 6.1. Observe that in the Nested reformulation, we interleave the product (6.2) into

$$\underbrace{\big(\underbrace{(\cdots\underbrace{(\underbrace{(y_{\sigma(1)})}_{=z_1}\, y_{\sigma(2)})}_{\substack{=z_2\\ \ddots}}\cdots y_{\sigma(p-1)})}_{=z_{p-1}}\, y_{\sigma(p)}\big)}_{=z_p},$$

where $z_1 = y_{\sigma(1)}$ and $z_i = y_{\sigma(i)}z_{i-1}$ for each $i = 2, \ldots, p$. We note that the u variables are used in Constraints (6.14)–(6.16), i.e., the McCormick relaxation constraints for the binary multiplication of $y_{\sigma(i)}z_{i-1}$ for each $i = 2, \ldots, p$. We know that the number of bits required to represent $y_{\sigma(i)}$ is equal to $n^y_{\sigma(i)}$ for each $i = 1, 2, \ldots, p$. We also know that $z_{i-1} \leqslant \prod_{j=1}^{i-1} \bar{y}_{\sigma(j)}$ for each $i = 2, \ldots, p$. Hence, the number of bits required to represent z_{i-1} is $\lfloor \sum_{j=1}^{i-1} \log_2 \left(\bar{y}_{\sigma(j)}\right) \rfloor + 1$ for each $i = 2, \ldots, p$. Further, the total number of u variables in the Nested reformulation is

$$\sum_{i=2}^{p} \left(n^y_{\sigma(i)} \left(\left\lfloor \sum_{j=1}^{i-1} \log_2 \left(\bar{y}_{\sigma(j)}\right) \right\rfloor + 1 \right) \right).$$

Similarly, observe from Constraints (6.17)–(6.21) that in order to compute z_i, some v variables and c variables should be generated for each $i = 2, \ldots, p$. Specifically, when computing z_i, the number of bits required to represent z_i is the number of v variables (or the number of c variables). Hence, the total number of c and v variables is

$$2\sum_{i=2}^{p}\left(\left\lfloor\sum_{j=1}^{i}\log_2\left(\bar{y}_{\sigma(j)}\right)\right\rfloor+1\right). \qquad \square$$

Proof of Proposition 6.2. Observe that in the Altogether reformulation, we calculate the product (6.2) as a whole, i.e.,

$$z=\prod_{i\in I} y_i.$$

We note that the u variables are used in Constraints (6.27)–(6.28), i.e., the McCormick relaxation constraints for the binary multiplication of $\prod_{i\in I} y_i$. We know that $z \leqslant \prod_{j\in I}\bar{y}_i$ and that $\bar{y}_i$ requires n_i^y bits to be represented for each $i = 1,\ldots,p$. Therefore, the total number of u variables in the Altogether reformulation is

$$\prod_{i\in I} n_i^y.$$

Similarly, observe from Constraints (6.29)–(6.32) that in order to compute z, some v and c variables should be generated. Specifically, when computing z, the number of bits required to represent z is the number of v variables (or the number of c variables). Therefore, the total number of c and v variables is

$$2\left(\left\lfloor\sum_{i\in I}\log_2\left(\bar{y}_i\right)\right\rfloor+1\right)=2n^z.$$

$\square$

E2: Extreme Examples

In this section, we provide the model of each extreme example in '.lp' format. We do not provide the objective function of the models as it is simply the multiplication of all y-variables. We first provide the model for the extreme example with $p = 15$. In this model, $C1$ to $C15$ are equations used to represent y-variables.

C1: $y_1 + 8x_1 + 4x_2 + 9x_7 + 4x_8 + x_9 + x_{10} + 6x_{12} - 4x_{13} - 5x_{17} = 18$

C2: $y_2 - 3x_1 + 6x_6 + 10x_7 + 9x_8 + 10x_9 - x_{10} - 9x_{11} - 3x_{14} + 7x_{16} - 2x_{19} = 20$

C3: $y_3 + 6x_1 - 7x_2 + 10x_4 - 2x_5 - 10x_6 - 4x_8 - 5x_9 + 9x_{11} - 2x_{12} + 7x_{14} + 7x_{15} - 7x_{17} - 9x_{19} + x_{20} = 34$

C4: $y_4 + 4x_3 - 10x_5 - 10x_7 + 4x_8 + 3x_9 - 5x_{12} + 10x_{13} + 2x_{16} - 10x_{17} + 8x_{18} - 3x_{19} = 23$

C5: $y_5 - 8x_1 - 3x_3 - 5x_4 + 3x_5 + 6x_{10} + x_{12} - 4x_{15} + 10x_{16} + 10x_{18} + 6x_{19} = 35$

C6: $y_6 + 9x_3 - 7x_4 - 10x_6 - 6x_8 + 7x_9 + 8x_{12} - 3x_{20} = 10$

C7: $y_7 + x_1 - 7x_4 - 4x_{10} + 5x_{15} - 2x_{17} - 3x_{19} = 13$

C8: $y_8 + 7x_1 - 3x_2 + 4x_3 + 9x_5 - 2x_7 + 7x_8 - 7x_9 - 6x_{11} - 2x_{13} - 10x_{16} + 3x_{17} - 7x_{18} = 16$

C9: $y_9 - 10x_1 + 5x_2 - 8x_4 - 6x_6 - 4x_9 + 4x_{10} - 6x_{12} + 7x_{13} + x_{16} + 10x_{17} - 10x_{18} + 9x_{19} = 13$

C10: $y_{10} + 4x_1 - 7x_3 + 4x_6 - 6x_7 + 4x_{11} - 10x_{12} + 2x_{14} + x_{15} - 3x_{17} - 9x_{19} + 10x_{20} = 17$

C11: $y_{11} - 9x_1 - 7x_2 + 9x_4 - 6x_5 - 10x_7 - 4x_{11} + 2x_{17} + 7x_{19} = 17$

C12: $y_{12} - 3x_6 + 2x_7 + x_9 + 2x_{12} + 5x_{16} + 7x_{19} = 17$

C13: $y_{13} - 2x_3 + 9x_4 + x_6 + 8x_7 + 2x_9 + 6x_{12} - x_{15} - 4x_{16} + x_{18} + 9x_{19} - 9x_{20} = 25$

C14: $y_{14} - 5x_1 + 4x_3 + 4x_5 - 2x_8 - 5x_9 + 4x_{10} - 8x_{11} - 9x_{13} - 9x_{14} - 4x_{15} - 10x_{16} + 7x_{17} - 9x_{18} - 2x_{20} = 1$

C15: $y_{15} + x_2 - x_8 - 6x_9 - 5x_{10} + 9x_{11} + 7x_{14} + 2x_{15} - x_{16} + 5x_{20} = 24$

C16: $11x_1 + 11x_5 + 27x_{10} + 4x_{11} + 22x_{12} + 22x_{15} + 24x_{19} + 28x_{20} \leqslant 32$

C17: $27x_2 + 21x_3 + 13x_5 + 7x_7 + 15x_8 + 19x_{10} + 25x_{11} + 3x_{12} + 4x_{13} + 5x_{16} + 15x_{17} \leqslant 27$

C18: $8x_2 + 24x_3 + 15x_7 + 21x_{11} + 16x_{13} + 17x_{14} + 22x_{16} + 15x_{18} + 8x_{19} \leqslant 60$

C19: $26x_1 + 2x_3 + 7x_4 + 16x_5 + 26x_6 + 23x_8 + 17x_9 + 24x_{11} + 30x_{13} + 29x_{14} + 18x_{17} + 15x_{18} + 9x_{19} \leqslant 32$

C20: $21x_2 + 21x_3 + 28x_4 + 11x_5 + 20x_6 + 13x_7 + 5x_9 + x_{11} + 16x_{12} + 11x_{15} \leqslant 91$

C21: $16x_3 + 9x_4 + 4x_5 + 14x_6 + 20x_7 + 6x_9 + 8x_{15} + x_{16} \leqslant 80$

C22: $28x_3 + 17x_6 + 22x_7 + 7x_{11} + 27x_{14} + 29x_{15} + 11x_{17} \leqslant 52$

C23: $26x_1 + 10x_3 + 28x_5 + 26x_8 + 27x_{10} + 30x_{12} + 20x_{15} + 5x_{16} + 28x_{17} + 18x_{18} \leqslant 26$

C24: $2x_1 + 27x_2 + 30x_3 + 28x_5 + 29x_7 + 22x_9 + 7x_{10} + 29x_{12} + 4x_{13} + 24x_{19} + 4x_{20} \leqslant 27$

C25: $x_1 + 22x_3 + 9x_5 + 30x_{10} + 12x_{13} + 17x_{14} + 22x_{17} + 5x_{18} + 30x_{19} + 19x_{20} \leqslant 89$

$x_i \in \{0,1\} \quad \forall i \in \{1,2,\cdots,20\}$

We now provide the model for the extreme example with $p = 20$. In this model, $C1$ to $C20$ are equations used to represent y-variables.

C1: $y_1 - 10x_1 + 3x_2 + 7x_3 - 8x_4 - 7x_6 - x_7 - 9x_9 - 2x_{10} + 2x_{11} - 10x_{13} - 8x_{14} + 3x_{15} + 8x_{16} + 10x_{17} + 3x_{18} + 10x_{19} - 6x_{20} = 25$

C2: $y_2 + 5x_2 + 6x_5 + 5x_6 - 10x_7 + 6x_{11} + 6x_{12} - 9x_{14} + 6x_{15} + 4x_{17} - 6x_{18} + 8x_{19} - 8x_{20} = 21$

C3: $y_3 + x_3 - 3x_4 + 4x_6 - 7x_8 - 10x_9 + 6x_{10} + 6x_{11} + 4x_{19} + 4x_{20} = 17$

C4: $y_4 + 10x_4 - 3x_7 - 3x_{11} - 8x_{13} + 3x_{14} - 9x_{19} = 16$

C5: $y_5 - 10x_2 + 7x_3 - 8x_5 - 5x_6 - 5x_7 + 2x_8 - 7x_{10} - 4x_{13} + 2x_{17} + 5x_{18} + 7x_{19} + 6x_{20} = 24$

C6: $y_6 - 9x_1 - 8x_3 - 4x_4 - 5x_8 - 8x_{10} + 6x_{13} + 9x_{15} - 3x_{17} - 4x_{19} = 9$

C7: $y_7 - 7x_2 + 5x_8 - 2x_{10} - 7x_{12} - 9x_{15} - 4x_{16} - 8x_{18} + 2x_{19} - 2x_{20} = 14$

C8: $y_8 + 5x_2 + 2x_8 + 5x_{10} - 10x_{11} - 6x_{12} - 2x_{15} + 7x_{16} + 3x_{17} - 6x_{18} - 8x_{19} + 8x_{20} = 19$

C9: $y_9 - 8x_9 + x_{10} - 8x_{15} + 10x_{18} + 8x_{19} + 2x_{20} = 21$

C10: $y_{10} + x_2 - 3x_4 + 4x_5 + 8x_6 + 5x_7 + 6x_8 - 10x_{10} - x_{12} + 7x_{13} - 4x_{15} + 2x_{16} + 8x_{19} - 2x_{20} = 24$

C11: $y_{11} + 9x_1 - 2x_3 + 2x_4 - 8x_5 + 7x_{10} - 3x_{11} - 6x_{14} - 6x_{17} - 8x_{18} = 18$

C12: $y_{12} - 2x_3 + 3x_4 - 8x_5 - x_9 - 8x_{10} + 10x_{13} - x_{14} - 3x_{15} - 5x_{17} - 9x_{19} - 9x_{20} = 20$

C13: $y_{13} + 3x_3 - 6x_4 + 2x_5 + 4x_7 - 5x_8 + 2x_{14} - 4x_{15} - 2x_{16} = 19$

C14: $y_{14} + 8x_5 + 7x_6 - 2x_8 - 6x_{13} + 5x_{17} - 4x_{18} - 2x_{19} - 5x_{20} = 18$

C15: $y_{15} + 6x_4 - 6x_6 + 7x_8 + 5x_{11} - 2x_{17} + 10x_{18} - 9x_{19} + 2x_{20} = 31$

C16: $y_{16} - 6x_1 + 4x_2 - 6x_3 + 9x_4 - 5x_5 + 9x_{12} + 7x_{15} - 8x_{16} + 2x_{17} + 7x_{18} + 10x_{19} + 7x_{20} = 28$

C17: $y_{17} + 6x_1 - 10x_2 - 10x_3 - 5x_6 + 2x_{14} - 10x_{16} - 10x_{19} - 10x_{20} = 9$

C18: $y_{18} - 4x_2 - 5x_5 + x_6 + 7x_7 - 8x_8 + 5x_9 + 6x_{10} - x_{11} - 8x_{12} - 5x_{14} + 3x_{15} + 3x_{18} - 6x_{20} = 16$

C19: $y_{19} + 10x_1 + x_5 - x_6 + 4x_8 + 9x_9 + 10x_{12} + x_{14} - 5x_{17} - 9x_{18} + 4x_{20} = 27$

C20: $y_{20} - 10x_1 + 10x_4 + x_9 + 4x_{10} + 9x_{12} + 4x_{13} + 4x_{16} - 5x_{17} + 10x_{20} = 38$

C21: $8x_1 + 4x_2 + 13x_4 + 19x_6 + 19x_9 + 21x_{10} + 8x_{11} + 29x_{14} + 11x_{15} + 25x_{16} + 3x_{17} \leqslant 22$

C22: $26x_2 + 30x_3 + 19x_4 + 2x_{12} + 3x_{13} + 3x_{17} + 29x_{19} \leqslant 44$

C23: $14x_1 + 3x_3 + 13x_4 + 10x_5 + 9x_8 + 19x_9 + 2x_{10} + 9x_{11} + 5x_{13} + 24x_{15} + 22x_{16} + 27x_{17} + 19x_{18} + 7x_{19} \leqslant 96$

C24: $27x_2 + 26x_4 + 27x_5 + 15x_{10} + 21x_{13} + 4x_{14} + 16x_{15} + 25x_{19} \leqslant 49$

C25: $15x_3+13x_4+10x_5+7x_7+24x_8+12x_9+10x_{11}+28x_{13}+18x_{14}+10x_{15}+10x_{17}+14x_{19} \leqslant 42$

C26: $16x_1 + 12x_4 + 13x_6 + 18x_7 + 3x_8 + x_9 + 4x_{10} + x_{13} + 28x_{15} + 24x_{16} + 19x_{17} + 9x_{20} \leqslant 54$

C27: $15x_1 + 25x_2 + 12x_7 + 3x_9 + 2x_{13} + 21x_{14} + 27x_{15} + 23x_{17} \leqslant 21$

C28: $20x_2 + 20x_3 + 16x_7 + 17x_8 + 26x_{11} + 24x_{12} + 10x_{18} \leqslant 53$

C29: $22x_3 + 22x_6 + 25x_7 + 3x_8 + 21x_9 + x_{11} + 15x_{14} + 12x_{15} + 24x_{18} + 4x_{19} \leqslant 89$

C30: $16x_1 + 27x_2 + 30x_5 + 9x_8 + 27x_{10} + 16x_{12} + 15x_{14} + 17x_{15} + 7x_{16} + 11x_{18} + 16x_{20} \leqslant 35$

$$x_i \in \{0,1\} \quad \forall i \in \{1, 2, \cdots, 20\}$$

E3: Instance Generator

We generate a total of 900 instances for the computational study. From the total of 900 instances, 300 are pure binary instances (which will be used for both minimization and maximization forms), 300 are pure continuous instances for the minimization form, and 300 are pure continuous instances for the maximization form. We generate the binary instances in such a way that they become challenging to solve both in the form of minimization or maximization. Note that generating pure continuous instances using the same settings that we employ for generating binary instances often results in instances that can be solved in just a fraction of a second by GRB, and that cannot be used for making any meaningful comparisons. Hence, we make some changes in the settings when creating continuous instances. Moreover, as mentioned in the introduction, continuous instances in maximization form can be solved in polynomial time (while mMPs remain NP-hard even in pure continuous form). Therefore, our pure continuous instances in the maximization form are larger than our pure continuous instances in the minimization form.

Specifically, pure binary instances are divided into three classes based on their value of $p \in \{2, 3, 4\}$. Each class contains 20 subclasses of instances based on the dimensions of the matrix $A_{m \times n}$, and each subclass contains 5 randomly generated instances. We consider $n \in \{100, 200, 300, 400, 500\}$ and $m = \alpha n$, where $\alpha \in \{0.5, 1, 1.5, 2\}$. For example, our smallest subclass is 100×50 with $n = 100$ x-variables and $m = 50$ constraints (related to x-variables), i.e., $\alpha = 0.5$, and our largest subclass is 500×1000 with $n = 500$ x-variables and $m = 1000$ constraints (related to x-variables), i.e., $\alpha = 2$. The sparsity of matrix A is set to 50%, i.e. $s_A := 0.5$, and the entries of matrix A are randomly drawn from the discrete uniform distribution $[1, 30]$. The components of vector $\boldsymbol{b}$ are randomly drawn from the discrete uniform distribution $[ns_A, 10ns_A]$, where ns_A is the expected number of non-zero elements in each row of matrix A. The sparsity of matrix D is also set to 50%, and its entries in row $i \in I$ are drawn randomly from the discrete uniform distribution $[-10i, 10i]$. Note that each row represents a linear function defining a y-variable. By generating the entries of D in that way, different y-variables require different numbers of bits to be represented. To ensure that the instances are non-trivial in the form of both minimization and maximization, we make the y-variables highly conflicting. Specifically, if we decide to assign a non-zero value to the entry located in row $i \in I$ and column $j \in \{1, \ldots, n\}$ of matrix D, we first count how many times column j has taken positive and negative values in previously generated rows of D. If the number of positives (negatives) is larger than the number of negatives (positives), then we make sure that the value of the entry located in row $i \in I$ and column $j \in \{1, \ldots, n\}$ is negative (positive). In the case of a tie, no restriction on the sign is imposed. Finally, to assure that every objective function takes a positive value, we first assume that all the elements of vector $\boldsymbol{d}$ are zero and solve an optimization problem for each $i \in I$ to compute the minimum possible value for y_i, denoted by L_i. We then randomly draw the components of $\boldsymbol{d}$ from the discrete uniform distribution $[|L_i| + 1, |L_i| + 10]$.

For pure continuous instances in minimization form, we generate 300 instances similar to the procedure described for generating pure binary instances. The only difference is that the entries of matrix A for pure continuous instances are randomly drawn from the discrete uniform distribution $[-30, 30]$ (rather than $[1, 30]$). This makes the feasible set of the instances larger and consequently instances are expected to become more challenging to solve. We also generate 300 instances for the maximization form similarly to the procedure described for minimization. The only difference is that the dimensions of matrix $A_{m\times n}$ are set to larger values. Specifically, we use $n \in \{400, 800, 1200, 1600, 2000\}$ and $m = \alpha n$, where $\alpha \in \{0.5, 1, 1.5, 2\}$ for pure continuous maximization instances. This implies that the smallest subclass of the pure continuous instance in maximization form has 400 x-variables and 200 constraints (related to x-variables). Similarly, the largest subclass of the pure continuous instance in maximization form has 2000 x-variables and 4000 constraints (related to x-variables).

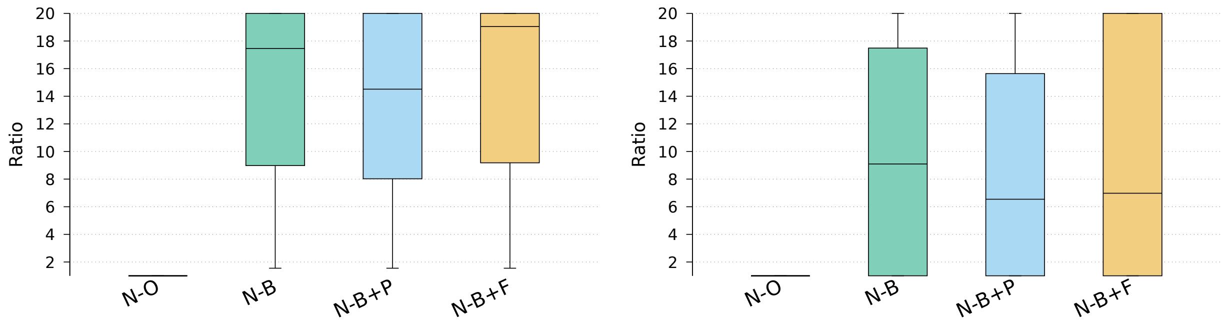
E4: Detailed Experimental Results

In this section, we provide the details of all our four experiments outlined in Section 6.5. Throughout this section, we use two types of charts for comparing the performance of different algorithms/solvers, one for time comparison and the other for solution quality comparison.

For solution time comparisons, we use a specific *boxplot* in which on the horizontal axis different algorithms are listed, and on the vertical axis the run time ratio is provided. Specifically, for constructing a boxplot, for each instance and for each solution approach, we need to compute the ratio of the run time of the approach to the minimum of the run times of all approaches for that specific instance. Hence, the closer the ratio is to one, the better the approach. Note that we do not report the solutions times explicitly when describing the details of our experiments in this section. This is because interested readers may refer to the 185-page supplementary PDF document (or its corresponding CSV file) available at https://github.com/paymanghasemi/Multiplicative-Programs-by-Binary-encoding-the-Multiplication-Operation to see the details for every instance and run. However, for convenience, at the end of this section, we report the averages solution times (in seconds) of our best solution approaches compared to GRB SOCP or GRB Nonconvex for different subclasses of instances in the form of tables.

For solution quality comparisons, we employ *performance profiles* as they provide more details compared to a boxplot [52]. The reason that we need to show more details to reader is that GRB SOCP or GRB Nonconvex can face numerical issues and report incorrect optimal solutions when solving a multiplicative program. We measure the solution quality of an approach for a multiplicative program as the gap between the objective value reported by the approach and the best objective value reported by all algorithms available on the same chart. We refer to this gap as the *best gap*, which helps identifying the cases where GRB SOCP or GRB Nonconvex face numerical issues.

In light of the above, the solution quality performance profiles present cumulative distribution functions for a set of solution approaches being compared with respect to their best gaps. The performance profiles show the best gaps on the horizontal axis and, on the vertical axis, for each approach, show the percentage of instances with a best gap that is smaller than or equal to the best gap on the horizontal axis. This means that values in the upper left-hand corner of the graph indicate the best performance.



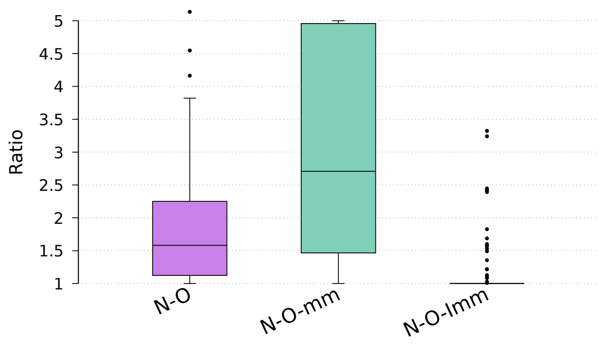
(a) Minimization

(b) Maximization

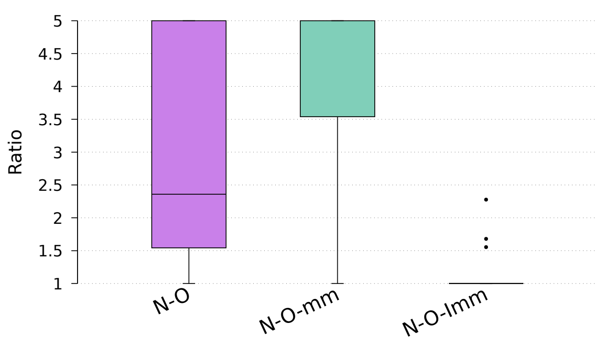
Figure E4.1: Solution time comparison between the proposed search mechanisms for pure binary instances with $p = 2$

E4.1: Experiment 1 Identifying the Best Search Mechanism

In this experiment, we intend to find which of our proposed search mechanisms, including One-shot, Bitwise, Bitwise + P-cut, and Bitwise + F-cut, is the fastest in solving multiplicative programs. We limit the focus of this experiment to instances with $p = 2$ as the Altogether and Nested formulations are the same for such instances. Figure E4.1 shows the solution time comparison of different search mechanisms. From this figure, we can observe that for both minimization and maximization instances, One-shot is the fastest search mechanism. The median of the One shot approach solution times is 6 to 20 times smaller than the solution times of other search mechanisms. This is not surprising as Bitwise is more suitable for resolving the curse of multiplication (and not for instances with small values of p). In the remaining experiments, we use the One-shot search mechanism whenever calling our proposed method. As an aside, we note that from Figure E4.1, we also observe that Bitwise + P-cut performs significantly better than Bitwise and Bitwise + F-cut. Therefore, users may want to use Bitwise + P-cut if they want to employ the Bitwise search mechanism.



(a) $p = 2$



(b) $p = 3$

Figure E4.2: Solution time comparison between different enhancement settings for pure binary minimization instances

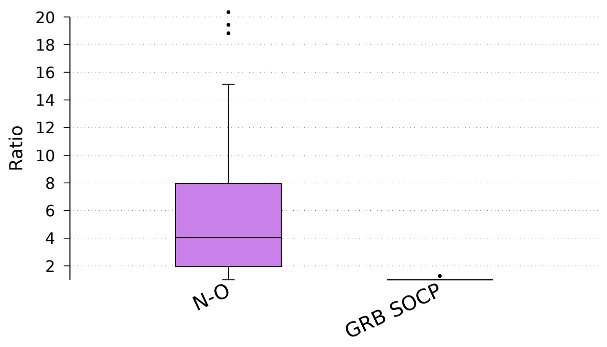
E4.2: Experiment 2 Identifying the Best Enhancement Technique for Minimization Instances

By fixing One-shot as the default search mechanism (based on the results of Experiment 1), in this experiment, we test our proposed warm-start enhancements for minimum multiplicative programs. Figure E4.2 provides the solution time comparison between the three enhancement settings, i.e., no warm-start, Min-Min, and Indirect Min-Min, for pure binary multiplicative instances with $p \in \{2, 3\}$. We later show in Experiment 3 that the Altogether reformulation does not perform well. Therefore, in this experiment, we use the Nested reformulation to deal with instances where $p = 3$. We can observe that Indirect Min-Min significantly decreases (by a factor of 1.5 to 5) the solution time for almost all instances. As an aside, we also observe that the Min-Min warm-start setting increases the overall solution time compared to using no warm-start as it is too time-consuming to solve the optimization problem for the Min-Min strategy. In the remaining experiments, the Indirect Min-Min setting will be active when solving minimization instances.

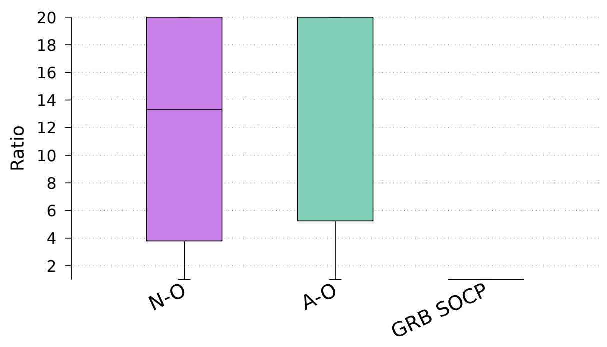
E4.3: Experiment 3 Comparing Different Solution Methods for Binary Instances

In this experiment, we compare the overall performance of our proposed solution approaches, including the Nested and Altogether reformulations (when One-shot and Indirect Min-Min are set as the default), with the performance of GRB SOCP and GRB Nonconvex on pure binary instances with $p \in \{2, 3, 4\}$. Figure E4.3 provides the solution time comparison on the pure binary maximum multiplicative instances. In Figure E4.3a, we can observe that GRB SOCP dominates our proposed solution approaches for any $p \in \{2, 3, 4\}$. Similarly, in Figures E4.3b and E4.3c, we can observe that the Nested reformulation performs better than the Altogether reformulation. However, it is evident that none of them are capable of competing with GRB SOCP.

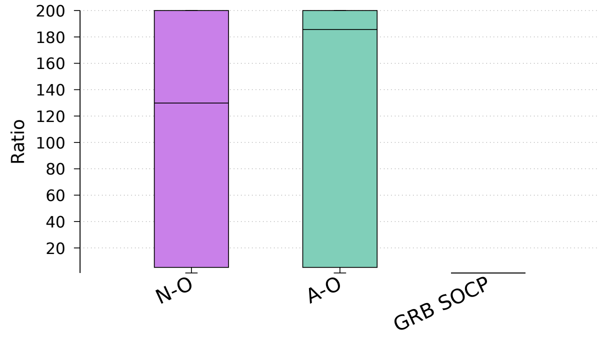
Figure E4.4 provides the solution time comparison on pure binary minimum multiplicative instances. We observe that our proposed Nested reformulation outperforms GRB Nonconvex for any $p \in \{2, 3, 4\}$. From Figures E4.4b and E4.4c, we observe that, for instances



(a) $p = 2$



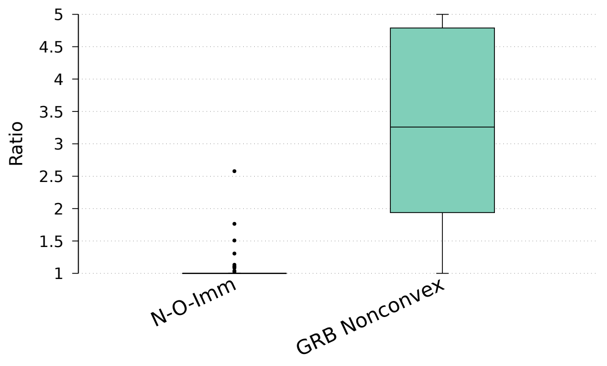
(b) $p = 3$



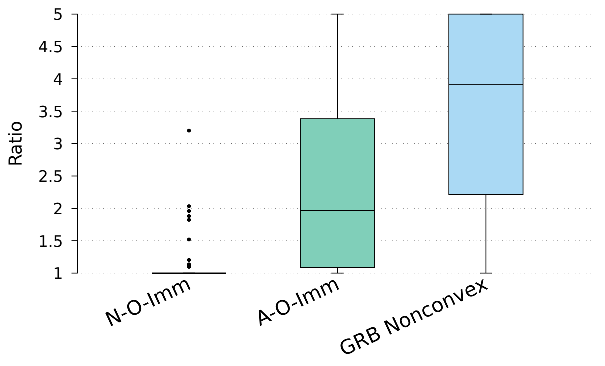
(c) $p = 4$

Figure E4.3: Solution time comparison between the proposed algorithms and GRB SOCP for binary maximization instances

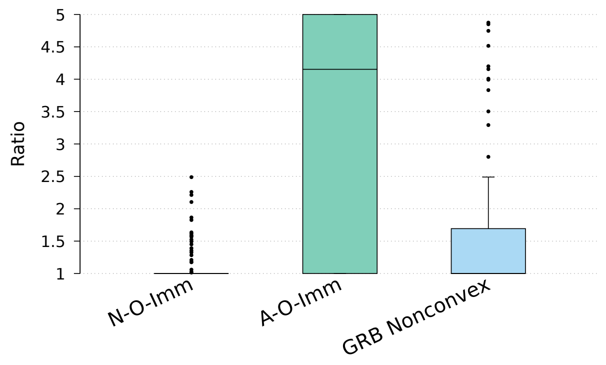
with $p \in \{3, 4\}$, the Nested reformulation performs significantly better than the Altogether reformulation. Further, we observe that even the Altogether reformulation outperforms GRB Nonconvex on the binary minimum instances with $p = 3$. However, we observe from Figure E4.4c that GRB Nonconvex performs better than the Altogether reformulation. Overall, due to the poor performance of the Altogether reformulation, we set the Nested reformulation as the default setting for the last experiment.



(a) $p = 2$



(b) $p = 3$



(c) $p = 4$

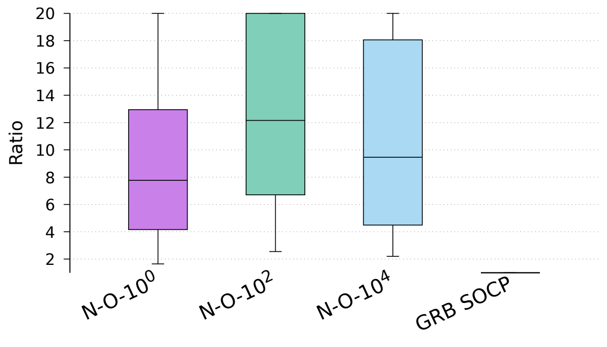
Figure E4.4: Solution time comparison between the proposed algorithms and GRB Nonconvex for binary minimization instances

E4.4: Experiment 4 Comparing Different Solution Methods for Continuous Instances

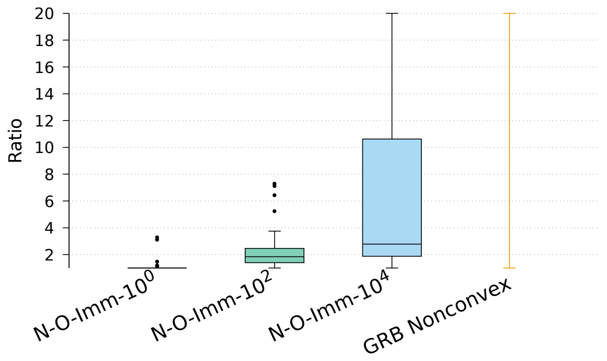
In this experiment, we compare the overall performance of our proposed Nested reformulation (when One-shot and Indirect Min-Min are set as the default) with the performance of GRB SOCP and GRB Nonconvex on pure continuous instances with $p \in \{2, 3, 4\}$. We note that for continuous instances, our approach can be used for only generating approximations (with any desirable level of precision) based on the discussion in Section 6.3.4. Hence, we conduct our experiments for three scenarios where 0, 2, and 4 digits after the decimal point are considered for each continuous y-variable, i.e., $\beta \in \{0, 2, 4\}$. In other words, we consider three multipliers $\{10^0, 10^2, 10^4\}$ for transforming the continuous y-variables into integers.

First, we report our results for only instances with $p = 2$. We note that for some continuous instances, GRB (including SOCP, Nonconvex, and its MIP solvers) reports errors, e.g., "unable to satisfy optimality tolerances", and/or terminates early. This was not observed for pure binary instances. While such issues can possibly be resolved by changing/tuning internal parameters of GRB (as described in its manual), we did not change them as it is not clear what the most efficient way of tuning is. Therefore, to have a fair comparison, we simply remove such instances in our comparisons in this section and report the number of removed instances wherever appropriate.

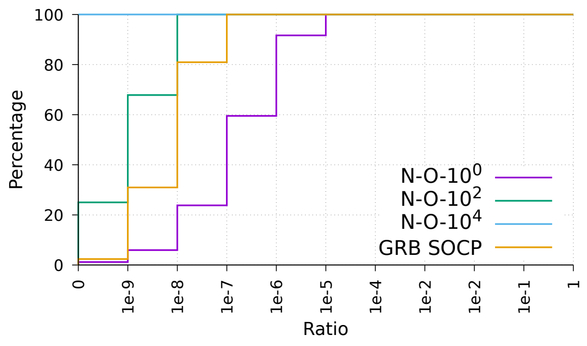
Figure E4.5 provides the performance comparison between the solution approaches for pure continuous instances with $p = 2$. From the continuous maximization instances with $p = 2$, a total of 16 instances are removed (due to errors or early termination) before generating the figure. However, no instances are removed for the minimization instances. For the maximization instances, we observe from Figure E4.5a that GRB SOCP dominates our proposed algorithm with respect to solution time. However, one interesting observation is that the solutions reported by our approach with multipliers of 10^2 and 10^4 have a better quality than those reported by GRB SOCP (although the optimality gap tolerance of GRB SOCP is set to zero). This again shows the power of the proposed solution method in handling numerical issues. For the maximization instances, we observe from Figure E4.5b that our proposed algorithm outperforms GRB Nonconvex with respect to solution time. However, we observe from Figure E4.5d that GRB Nonconvex has obtained better quality solutions. Overall, we observe that mMPs and MMPs have completely opposite performance



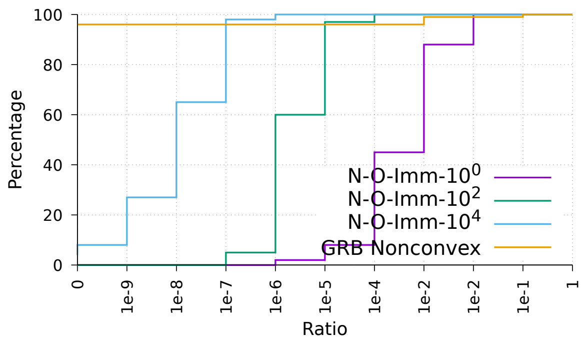
(a) Solution time comparison for maximization



(b) Solution time comparison for minimization



(c) Solution quality comparison for maximization



(d) Solution quality comparison for minimization

Figure E4.5: Performance comparison between the proposed approach, GRB SOCP, and GRB Nonconvex for pure continuous instances with $p = 2$

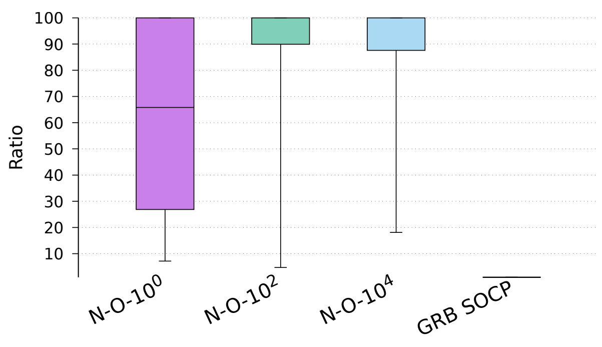
compared to our proposed approach. One reason for this could be the fact that continuous MMPs are polynomially solvable while continuous mMPs are NP-hard.

Figure E4.6 provides the performance comparison for pure continuous instances with $p = 3$. From the continuous maximization instances with $p = 3$, a total of 23 instances are removed (due to errors or early termination) before generating the figure. In total, only 2 instances are removed for the minimization instances. For the maximization instances, we observe from Figures E4.6a and E4.6c that GRB SOCP dominates our proposed algorithm with respect to both solution time and quality. However, for the minimization instances, we observe the opposite result. Specifically, Figures E4.6b and E4.6d show that our proposed algorithm outperforms GRB Nonconvex with respect to both solution time and quality (for the multiplier of 10^4).

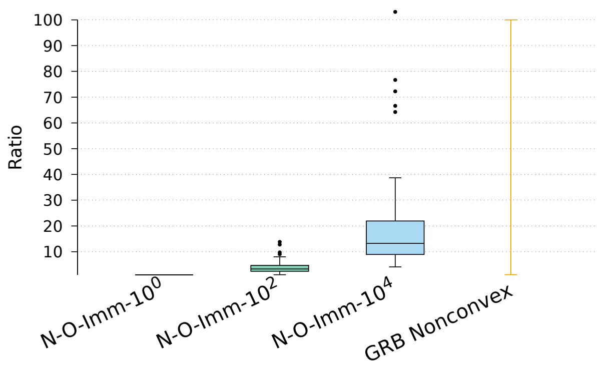
Figure E4.7 provides the performance comparison for pure continuous instances with $p = 4$. From the continuous maximization instances with $p = 4$, a total of 8 instances are removed (due to errors or early termination) before generating the figure. In total, 14 instances are removed for the minimization instances. For the maximization instances, we again observe from Figure E4.7a and E4.7c that GRB SOCP dominates our proposed algorithm with respect to both solution time and quality. However, for the minimization instances, we observe from Figures E4.7b and E4.7d that our proposed algorithm outperforms GRB Nonconvex with respect to both solution time and quality (for the multiplier of 10^0). We also observe that as the multiplier increases, we do not necessarily see an improvement in the quality of solutions reported. This is because when $p = 4$, the instances become larger and more difficult to solve. Hence, many instances with a multiplier of 10^4 cannot be solved to optimality within the time limit.

E4.5: Report Average Solution Times

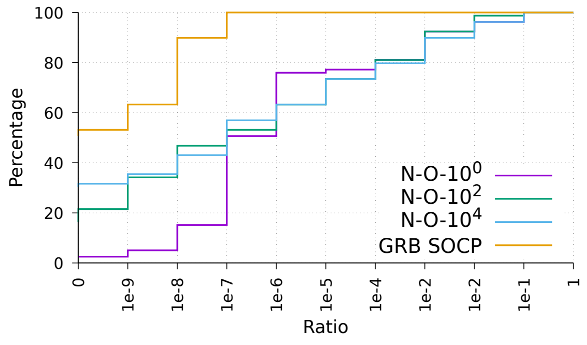
In the previous parts of the paper, we have not provided any information about the actual solution times (in seconds). So, the propose of this section is to report such



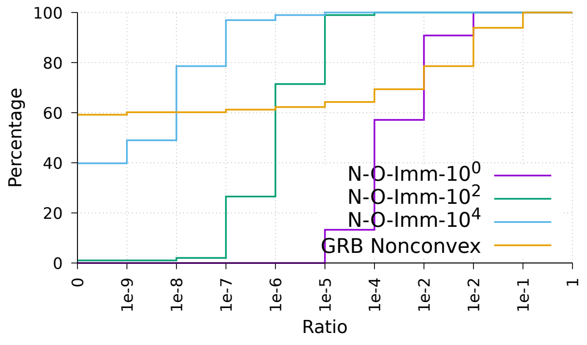
(a) Solution time comparison for maximization



(b) Solution time comparison for minimization

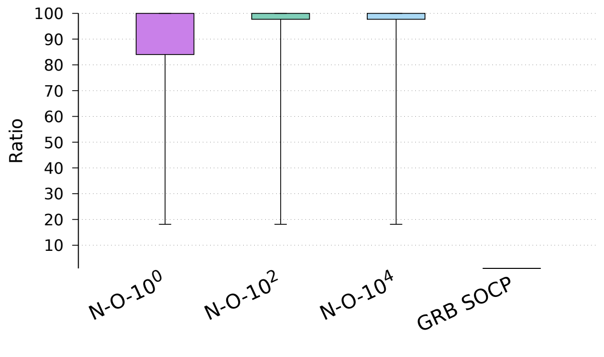


(c) Solution quality comparison for maximization

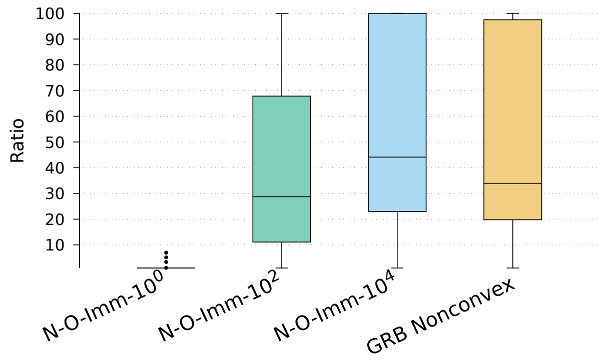


(d) Solution quality comparison for minimization

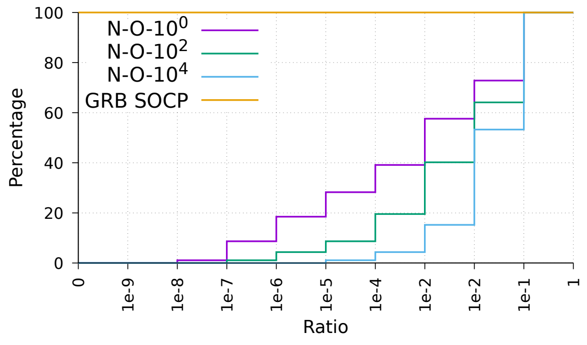
Figure E4.6: Performance comparison between the proposed approach, GRB SOCP, and GRB Nonconvex for pure continuous instances with $p = 3$



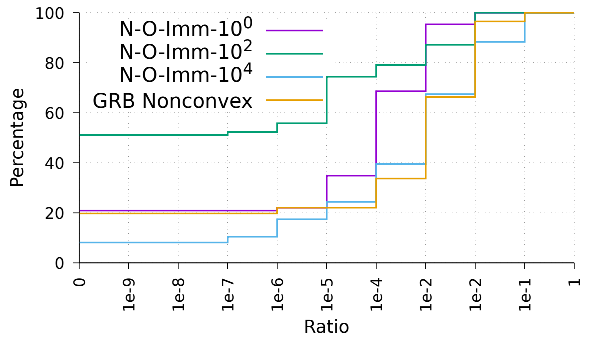
(a) Solution time comparison for maximization



(b) Solution time comparison for minimization



(c) Solution quality comparison for maximization



(d) Solution quality comparison for minimization

Figure E4.7: Performance comparison between the proposed approach, GRB SOCP, and GRB Nonconvex for pure continuous instances with $p = 4$

information for interested readers. Note that based on Table 6.5, we know the best setting for our proposed solution approach for different types of instances. So, in this section, we report the average solution times (in seconds) per subclass for the best setting of our proposed solution approach compared to GRB (Nonconvex or SOCP). Specifically, Table E4.1 is for all binary instances (in both maximization and minimization forms); Table E4.2 is for continuous instances in the form of minimization; Table E4.3 is for continuous instances in the form of minimization. The figures reported in each table are averages over 5 instances. Each figure reported in the row labeled 'Avg' shows the average of the four figures above it. Recall that the imposed time limit is 3600 seconds for each method and instance in our computational study. So, in the tables, 3600 means that the instance is not solved to optimality.

Table E4.1: Average solution time (sec.) per subclass of pure binary instances

Subclass	$p = 2$				$p = 3$				$p = 4$			
	Max		Min		Max		Min		Max		Min	
$(n \times m)$	GRB SOCP	N-O	GRB Nonconvex	N-O-Imm	GRB SOCP	N-O	GRB Nonconvex	N-O-Imm	GRB SOCP	N-O	GRB Nonconvex	N-O-Imm
100 × 50	0.14	0.54	0.25	0.10	0.45	3.55	1.53	0.65	0.29	77.13	11.39	9.25
100 × 100	0.25	0.60	0.36	0.16	0.48	4.33	3.38	0.91	0.37	71.80	10.39	11.33
100 × 150	0.36	0.61	0.49	0.23	0.70	8.93	4.19	0.97	0.44	63.32	17	19.01
100 × 200	0.76	1.01	0.88	0.30	1.11	11.11	4.59	0.98	0.77	41.61	14.52	19.73
Avg	**0.38**	**0.69**	**0.49**	**0.20**	**0.69**	**6.98**	**3.42**	**0.88**	**0.47**	**63.46**	**13.33**	**14.83**
200 × 100	1.54	7.38	1.14	0.36	3.01	88.74	11.47	2.61	2.77	2,870.68	132.07	95.27
200 × 200	2.01	6.94	3.84	1.09	3.44	87.61	32.67	22.14	6.35	2,381.17	223.37	197.34
200 × 300	3.69	11.86	5.73	2.11	6.18	173.19	56.30	20.44	8.13	1,967.04	469.03	272.54
200 × 400	7.61	21.29	14.78	15.95	24.75	704.89	140.78	63.46	4.79	1,480.42	597.52	369.64
Avg	**3.71**	**11.87**	**6.37**	**4.88**	**9.34**	**263.61**	**60.30**	**27.16**	**5.51**	**2,174.83**	**355.50**	**233.70**
300 × 150	5.96	36.52	12.73	11.22	132.87	1,973.46	111.50	134.90	21.37	3,600	695.87	669.48
300 × 300	51.35	252.81	76.21	20.03	43.78	1,793.30	364.95	179.79	31.95	3,600	2,978.88	1,198.98
300 × 450	141.68	916.02	62.26	26.45	42.69	1,885.81	683.88	182.68	68.78	3,600	3,548.87	1,906.17
300 × 600	294.83	1,576.01	308.89	79.05	784.29	2,171.42	1,320.08	367.16	80.69	3,600	3,600	3,304.56
Avg	**123.45**	**695.34**	**115.02**	**34.19**	**250.91**	**1,956**	**620.10**	**216.14**	**50.70**	**3,600**	**2,705.91**	**1,769.80**
400 × 200	45.21	917.19	38.47	6.03	377.79	3,049.79	406.28	164.66	24.35	3,600	2,374.11	1,080.32
400 × 400	831.15	2,189.04	272.19	80.87	2,299.85	3,600	1,596.66	564.67	1,931.86	3,600	3,600	3,039.43
400 × 600	1,582.25	3,600	599.28	426.08	506.79	3,355.18	3,045.58	524.68	1,588.82	3,600	3,600	2,923.85
400 × 800	805.74	2,517.37	994.61	486.01	2,195.25	3,600	3,127.87	593.89	814.04	3,600	3,600	3,600
Avg	**816.09**	**2,305.90**	**476.14**	**249.75**	**1,344.92**	**3,401.24**	**2,044.10**	**461.98**	**1,089.77**	**3,600**	**3,293.53**	**2,660.90**
500 × 250	620.12	2,559.68	851.29	195.43	1,598.27	3,600	1,311.07	163.22	1,040.80	3,600	3,600	3,600
500 × 500	2,948.09	3,600	1,251.10	506.82	3,238.07	3,600	3,142.02	1,886.16	2,338.58	3,600	3,600	3,137.05
500 × 750	2,281.89	3,249.34	1,765.53	652.18	2,937.46	3,600	3,600	3,600	2,950.63	3,600	3,600	3,600
500 × 1000	3,600	3,600	2,565.95	816.81	3,013.91	3,600	3,600	3,600	3,140.03	3,600	3,600	3,600
Avg	**2,362.52**	**3,252.25**	**1,608.47**	**542.81**	**2,696.93**	**3,600**	**2,913.27**	**2,312.35**	**2,367.51**	**3,600**	**3,600**	**3,484.26**

Table E4.2: Average solution time (sec.) per subclass of continuous minimization instances

Subclass	$p = 2$		$p = 3$		$p = 4$	
($n \times m$)	GRB Nonconvex	N-O-Imm-10^0	GRB Nonconvex	N-O-Imm-10^0	GRB Nonconvex	N-O-Imm-10^0
100 × 50	37.33	0.23	220.88	1.45	998.77	44.78
100 × 100	5.69	0.26	68.38	1.23	2,261.40	19.21
100 × 150	5.07	0.31	72.50	1.08	1,430.73	26.11
100 × 200	5.55	0.37	118.20	1.66	620.05	10.75
Avg	**13.41**	**0.29**	**119.99**	**1.35**	**1,327.74**	**25.21**
200 × 100	49.02	0.50	1,337.04	2.71	2,944.63	385.93
200 × 200	19.88	0.55	607.64	2.97	3,040.51	18.96
200 × 300	26.85	0.88	813.28	3.45	3,600	53.41
200 × 400	32.72	1.21	882.82	3.85	3,600	38.48
Avg	**32.12**	**0.79**	**910.20**	**3.25**	**3,296.28**	**124.19**
300 × 150	92.34	1.13	3,590.85	5.05	1,479.46	92.66
300 × 300	62.85	1.50	2,849.71	5.55	3,600	37.87
300 × 450	82.94	1.87	3,363.13	9	3,600	60.60
300 × 600	88.51	3.18	3,329.60	8.75	3,600	277.01
Avg	**81.66**	**1.92**	**3,283.32**	**7.09**	**3,069.87**	**117.04**
400 × 200	1,359.81	1.89	3,600	8.17	2,123.66	122
400 × 400	184.88	2.57	3,131.84	14.08	3,600	112.85
400 × 600	212.07	4.73	3,600	12.77	3,600	122.28
400 × 800	234.05	4.14	3,600	15.61	3,600	630.45
Avg	**497.70**	**3.33**	**3,482.96**	**12.66**	**3,230.92**	**246.90**
500 × 250	3,600	2.56	3,600	12.61	3,485.64	231.24
500 × 500	256.90	4.94	3,600	17.63	3,600	840.50
500 × 750	331.01	6.06	3,600	26.87	2,885.45	492.56
500 × 1000	308.96	8.09	3,600	27.51	3,600	218.81
Avg	**1,124.22**	**5.41**	**3,600**	**21.16**	**3,392.77**	**445.78**

Table E4.3: Average solution time (sec.) per subclass of continuous maximization instances

Subclass	$p = 2$		$p = 3$		$p = 4$	
($n \times m$)	GRB SOCP	N-O-10^0	GRB SOCP	N-O-10^0	GRB SOCP	N-O-10^0
400 × 200	0.14	2.78	0.15	19.78	0.10	2,724.88
400 × 400	0.37	4.20	0.40	286.35	0.28	3,192.79
400 × 600	0.68	4.17	0.75	190.12	0.55	3,513.46
400 × 800	1.22	7.16	1.28	93.40	0.93	3,347.84
Avg	**0.60**	**4.58**	**0.64**	**147.41**	**0.46**	**3,194.74**
800 × 400	0.69	16.45	0.75	96.25	0.53	3,443.42
800 × 800	2.28	29.45	2.51	755.75	1.86	3,600
800 × 1600	9.43	25.07	10.78	506.14	8.04	3,600
800 × 1200	5.19	50.11	5.71	141.58	4.33	3,600
Avg	**4.39**	**30.27**	**4.94**	**374.93**	**3.69**	**3,560.85**
1200 × 600	2.13	38.30	2.58	1,429.86	1.84	3,600
1200 × 1200	8.73	44.51	9.40	1,566.67	8.26	2,925.10
1200 × 1800	19.97	141.72	22.20	2,324.84	22.43	3,600
1200 × 2400	38.44	160.13	40.59	2,374.15	35.69	3,600
Avg	**17.32**	**96.16**	**18.69**	**1,923.88**	**17.05**	**3,431.27**
1600 × 800	6.56	51.67	7.13	2,069.30	5.56	3,600
1600 × 1600	23.81	228.91	25.65	1,428.67	23.85	3,600
1600 × 2400	52.18	283.11	53.67	2,611.75	56.41	3,600
1600 × 3200	88.48	354.25	100.31	2,259.04	94.07	3,600
Avg	**42.76**	**229.49**	**46.69**	**2,092.19**	**44.97**	**3,600**
2000 × 1000	13.21	169.92	15.94	3,598.26	14	3,600
2000 × 2000	44.85	367.85	52.53	3,600	56.13	3,600
2000 × 3000	104.11	662.59	117.80	2,983.96	114.12	3,600
2000 × 4000	176.46	987.35	192.10	2,872.06	196.33	3,600
Avg	**84.66**	**546.93**	**94.59**	**3,263.57**	**95.15**	**3,600**