November 2019

Algorithms for Multi-Objective Mixed Integer Programming Problems

Alvaro Miguel Sierra Altamiranda
University of South Florida

Algorithms for Multi-Objective Mixed Integer Programming Problems

by

Alvaro Miguel Sierra Altamiranda

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:
November 1, 2019

Keywords: Optimization Over Efficient Frontier, Open-Source Software,
Machine Learning, Objective Projection, Nash Bargaining Solution

To my beautiful wife María, and my beloved family, for their unconditional support.

Acknowledgments

First and foremost, I would like to thank my advisor, Dr. Hadi Charkhgard, for his constant support throughout my Ph.D. journey. His continuous guidance, motivation, knowledge, and trustfulness were key to achieving all my research goals. It has been an absolute pleasure to work under his supervision, and to share knowledge and experiences with the members of his research group, the so-called Multi-Objective Optimization Laboratory at the University of South Florida (USF).

My sincere thanks also goes to Dr. Tapas K. Das for his consideration and encouragement to study a Ph.D. at USF. I am grateful to my friends in my native Colombia for the endless encouragement to pursue a Ph.D. in the United States. From them, I would like at least to mention Ernesto (Tico) Cantillo, who always showed true happiness for my professional and personal growth.

I would also like to thank all the talented individuals I had the pleasure to work with during my journey. Specially to Dr. Julien Martin, Dr. Shahab Mofidi, Dr. Iman Dayarian, Dr. Ali Eshragh, and Nader Rezaei. I would like 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 140G0118P0352.

I wish to thank my family, to whom I dedicate this dissertation. My parents, Alvaro and Mayra, who sacrificed everything to make sure I always received the best education. My hope is that I can be as you taught me to be, a better person every day. My sister Aileen and her husband Erik, who provided unconditional support. My grandmother Edith and aunt Yenny, who always kept me in their prayers. And finally, to my beautiful wife, Maria, whose endless support and unbelievable patience made of our journey the best of the adventures.

Table of Contents

List of Tables

List of Figures

Abstract

This thesis presents a total of 3 groups of contributions related to multi-objective optimization. The first group includes the development of a new algorithm and an open-source user-friendly package for optimization over the efficient set for bi-objective mixed integer linear programs. The second group includes an application of a special case of optimization over the efficient on conservation planning problems modeled with modern portfolio theory. Finally, the third group presents a machine learning framework to enhance criterion space search algorithms for multi-objective binary linear programming.

In the first group of contributions, this thesis presents the first (criterion space search) algorithm for optimizing a linear function over the set of efficient solutions of bi-objective mixed integer linear programs. The proposed algorithm is developed based on the triangle splitting method (Boland et al. [1]), which can find a full representation of the nondominated frontier of any bi-objective mixed integer linear program. The proposed algorithm is easy to understand and implement, and converges quickly to an optimal solution. An extensive computational study shows the efficacy of the algorithm. Is numerically shown in this thesis that the proposed algorithm can be used to quickly generate a provably high-quality approximate solution because it maintains a lower and an upper bound on the optimal value of the linear function at any point in time. Additionally, this thesis presents `OOESAlgorithm.jl`, a comprehensive julia package based on the proposed algorithm. The proposed package extends the first implementation of the algorithm by adding two main features: (a) in addition to CPLEX, the package allows employing any single-objective solver supported by MathProgBase.jl, for example, GLPK, CPLEX, and SCIP; (b) the package supports execution on multiple processors and is compatible with the JuMP modeling language. An extensive computational study shows the efficacy of the package and its features.

In the second group of contributions, this thesis presents a Nash bargaining solution approach for spatial conservation planning problems modeled with modern portfolio theory. The proposed modern portfolio optimization formulation corresponds to a spatial conservation planning problem involving two conflicting objectives: maximizing return and minimizing risk. A Nash bargaining solution approach is presented in this thesis to directly compute a desirable Pareto-optimal (nondominated) solution for the proposed bi-objective optimization formulation in natural resource management problems. Numerical examples in this thesis show that to directly compute a Nash bargaining solution, a Binary Quadratically Constrained Quadratic Program (BQCQP) can be solved. This thesis also shows that the proposed approach (implementable with commercial solvers such as CPLEX) can effectively solve the proposed BQCQP for much larger problems than previous approaches published in the ecological literature. The new approach expands considerably the applicability of such optimization methods to address real spatial conservation planning problems.

In the third group of contributions, this thesis investigates the possibility of improving the performance of multi-objective optimization solution approaches using machine learning techniques. Specifically, this thesis focus on multi-objective binary linear programs and employs one of the most effective and recently developed criterion space search algorithms, the so-called KSA, during our study. This algorithm computes all nondominated points of a problem with p objectives by searching on a projected criterion space, i.e., a $(p-1)$-dimensional criterion space. This thesis presents an effective and fast learning approach to identify on which projected space the KSA should work, and also presents several generic features that can be used in machine learning techniques for identifying the best projected space. Finally, a bi-objective optimization based heuristic for selecting the best subset of the features to overcome the issue of overfitting in learning is presented. Through an extensive computational study, the performance of the proposed learning approach is tested.

Chapter 1: Introduction

In this chapter, we present the research questions that will be answered through the contributions of this thesis. The research questions formulated in this chapter are all related to multi-objective mixed integer linear and nonlinear programming and optimization over the efficient set. These can be categorized as (1) algorithms questions (design and enhancement), (2) software question (package development), and (3) application question (in spatial conservation planning problems). The content of this chapter is organized as follows: In Section 1.1 we motivate and formulate our research questions. In Section 1.2 we present a list with our contributions, including published and submitted papers. It is worth mentioning that all the papers that represent a contribution to this thesis are co-authored, however, I am the first author in all of them. Finally, in Section 1.3 we briefly describe the content of this thesis.

1.1 Motivation

One of the focuses of this thesis is on Multi-Objective Mixed Integer Linear Programming (MOMILP) problems. It is well documented that problems which involve multiple objective and some (if not all) of their decision variables are integer are NP-hard, e.g., there is currently no algorithm that can solve any instance of this problem in polynomial time. Therefore, finding many or all *efficient solutions*, i.e., solutions in which improve the value of one objective is only possible if the value of at least another objective deteriorates, can take a considerable amount of time even for small instances. Moreover, even if many efficient solutions are found, the collection of them can be overwhelming for decision-makers.

Many studies in the literature of MOMILP mainly focus on computing a representation of the so-called *nondominated (Pareto) frontier*, e.g., a collection of the projections of the efficient solutions in the criterion space. However, the problem of choosing a preferred solution among the set of efficient solutions, even tho it has been documented for decades, has been lightly explored in integer programming. To the best of our knowledge, there are only a few studies related to optimization over the efficient set for multi-objective pure integer linear programming. Moreover, until the publication of one of the contributions of this thesis, there was no algorithm (exact or approximate) for optimization over the efficient set for MOMILP, not even for the special case of 2 objective functions. In lights of the above, we find the motivation to formulate our first research question based on the simplest case of MOMILP with 2 objective functions:

- Q1: How to design a new exact algorithm for the problem of optimization over the efficient set for bi-objective mixed integer linear programming?

Once an optimization algorithm is developed, its main purpose is to be used by any community (operations research, economy, geology, ecology, transportation science, etc.) in real-world applications. However, many of the available exact algorithms are not used because there are not open-source user-friendly implementations, available software often rely on commercial solvers, or they perform poorly in terms of computational time. In contrast, one of the most successful (approximate) algorithms for multi-objective optimization is the NSGA-II (Deb et al. [2]) because it is fast, easy to understand, easy to implement, and do not rely on commercial solvers, even tho it cannot guarantee that the solutions found are efficient solutions. In lights of the above, we find the motivation to formulate our second research question:

- Q2: How to develop an open-source user-friendly fast and flexible implementation for the algorithm that solves the problem of optimization over the efficient set for bi-objective mixed integer linear programming?

We mentioned above the ecological as an example of the multiple communities that can benefit from optimization algorithm implementations. One of the specific problems in ecology that draw our attention is spatial conservation planning. Many studies that formulate and solve spatial conservation planning problems as integer programs can be found in the literature. However, most of them fail to address the inherent uncertainty of these problems. An approach that alleviates this issue is to formulate the problem based on modern portfolio theory that propose 2 objective functions, maximizing the return, and minimizing the risk. But this formulation includes a quadratic objective function (the risk) with several bi-linear terms, i.e., multiplication of 2 decision variables.

To the best of our knowledge, there is only one study that models spatial conservation planning problems using modern portfolio theory (Eaton et al. [3]). This study relaxes the bi-linear terms using a McCormick Envelope, which creates at least 1 new decision variable and 3 new constraints for each bi-linear term. Evidently, this generates a series of memory issues and poor computational time performance when solving large scale conservation planning problems. Finally, they approximate the so-called Nash bargaining solution, which is a special case of optimization over the efficient set, to select their preferred solution. We formulate our third research question as a continuation of this research:

- Q3: How to formulate a Nash bargaining solution approach for the spatial conservation planning problem based on modern portfolio theory capable of handle large scale problems with minimal memory requirements?

Up to this point, our research questions are directly and indirectly related to the problem of optimization over the efficient set. However, for the fourth and final research question, we focus on finding the entire nondominated frontier of Multi-Objective Binary Linear Programming (MOBLP) problems. Moreover, we focus on exact criterion space search algorithms for MOBLP. Usually, these algorithms reduce the dimension of the criterion space by projecting one of the objective functions. How to select the objective function to project and how to do the projection are still open questions in MOBLP problems. Finally, we believe that both questions can be addressed by implementing machine learning techniques. In lights of the above, we find the motivation to formulate our last research question:

- Q4: How to develop a machine-learning-based framework that learns the best selection of objective function to project in MOBLP?

1.2 Contributions of the Thesis

We give answer to the formulated research questions throughout the body of this thesis. Our answers led to 4 major contributions: the publication of 2 journal articles, the submission of 1 journal article, and a final journal article in preparation (at the time of writing this thesis). Our contributions are listed below:

- P1: Sierra-Altamiranda, A., & Charkhgard, H. (2019). A New Exact Algorithm to Optimize a Linear Function over the Set of Efficient Solutions for Biobjective Mixed Integer Linear Programs. *INFORMS Journal on Computing.* https://doi.org/10.1287/ijoc.2018.0851.

- P2: Sierra-Altamiranda, A., & Charkhgard, H. (2019). OOESAlgorithm. jl: a julia package for optimizing a linear function over the set of efficient solutions for biobjective mixed integer linear programming. *International Transactions in Operational Research*. https://doi.org/10.1111/itor.12692.

- P3: Sierra-Altamiranda, A., Charkhgard, H., Eaton, M., Martin, J., Yurek, S., Udell, B., (2019). Spatial Conservation Planning Under Uncertainty Using Modern Portfolio Theory and Nash Bargaining Solution (Submitted to *Ecological Modelling*).

- P4: Sierra-Altamiranda, A., Charkhgard, H., Dayarian, I., Eshragh, A., Javadi, S. (2019). Learning to Project in Multi-Objective Binary Linear Programming (Submitted to *INFORMS Journal on Computing*).
  - Available at: http://www.optimization-online.org/DB_FILE/2019/01/7051.pdf.

1.2.1 Related Publications and Preprints

We present our paper P1 in Chapter 3, Section 3.3. In paper P1 we answer question Q1. We present a new criterion space search algorithm for optimize a linear function over the set of efficient solutions for bi-objective mixed integer linear programming. Our algorithm is easy to understand and easy to implement and can be used to generate a provably high-quality approximate solution. We detail the algorithm and its operations, and evaluate its performance in a comprehensive computational study.

We present our paper P2 in Chapter 3, Section 3.4. In paper P2 we answer question Q2. We develop `OOESAlgorithm.jl`, a comprehensive open-source user-friendly julia package for optimizing a linear function over the set of efficient solutions for bi-objective mixed integer linear programming. Our package supports several commercial and non-commercial solvers, multi-processing environments, and can be tuned to return the exact nondominated frontier of the original problem. All the features of the packages are evaluated in a comprehensive computational study.

We present our paper P3 in Chapter 4. In paper P3 we answer question Q3. We present a Nash bargaining solution approach for the spatial conservation planning problem based on modern portfolio theory. Our approach includes a transformation from the modern portfolio formulation which is a bi-objective binary quadratic problem to a single-objective quadratically constrained binary linear problem. Such transformation allows the use of second order cone programming (available in most commercial solvers) to solve the problem. A computational study shows that our formulation can generate (almost) optimal solutions for large scale spatial conservation planning problems.

We finally present our paper P4 in Chapter 5. In paper P4 we answer question Q4. We develop a machine learning approach to emulate the best selection of objective function to project in multi-objective binary linear programming. We create features describing an instance, and labels to classify the objective functions. Our approach learns a function from the features map of a training set and returns the label of the best objective function to project. We present a ordering approach for the objective functions and a bi-objective approach for the best subset selection of features. Finally, we present the efficiency of our approach enhancing the so-called KSA, i.e., a new exact algorithm for multi-objective binary linear programming, on a comprehensive computational study.

1.3 Outline of the Thesis

The remaining content of this thesis is organized as follows:

- In Chapter 2 we provide some important definitions and notation related to the general multi-objective optimization problems and single-objective nonlinear optimization problems.

- In Chapter 3 we present a new exact criterion space search algorithm for optimizing a linear function over the set of efficient solutions for bi-objective mixed integer linear programming. We present a comprehensive literature review of this problem, some preliminaries to understand the specifics of the problem, a detailed explanation of the algorithm, and a comprehensive computational experiment. We also present `OOESAlgorithm.jl`, a comprehensive open-source user–friendly julia package for solving the same problem. We evaluate the performance of the package with a second computational study. We finalize the chapter with some concluding remarks.

- In Chapter 4 we present a Nash bargaining solution approach for the spatial conservation planning problem based on modern portfolio theory. We present a comprehensive literature review on operations research applications in ecology, and specifically in spatial conservation planning. We present a modern portfolio theory formulation for spatial conservation planning. We present a Nash bargaining solution approach for this problem and apply some transformations to fit the formulation to a second order cone programming formulation. We present a computational study with simulated data and finalize with some concluding remarks.

- In Chapter 5 we present a machine learning approach that learns a function which emulates the best selection of objective function for projection in a reduced dimension of the criterion space of multi-objective binary linear programming problems. We highlight the features that describe a multi-objective binary linear programming instance. We present an ordering approach for the objective functions and a bi-objective optimization over the efficient set approach for the best subset selection of features. We present a comprehensive computational study using Assignment and Knapsack problems, and the KSA to solve the instances. We finalize with some concluding remarks.

- In Chapter 6 we provide a summary with the results of this thesis, give concluding remarks, and establish some paths for future research.

Chapter 2: Preliminaries

In this chapter, we present some preliminaries of the general problems that form the basis of the specific problems that are the focus of this thesis. From this point through the entire thesis, any array of variables or coefficients are column vectors and the labels for vectors are bolded.

2.1 Mixed-Integer Linear Programming

Many real world problems such as scheduling, supply chain management, and production planning can be formulated as *Mixed Integer Programs* (MIP). Moreover, if the objective function and constraints of the problem are linear functions, then the formulation is known as a *Mixed Integer Linear Program* (MILP). In a MILP, the feasible decision space is described by a set of affine constraints where some (if not all) decision variables are integer. Therefore, a MILP can be stated as follows:

$$\min_{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}} \; z(\boldsymbol{x}_I,\boldsymbol{x}_C), \tag{2.1}$$

where $\mathcal{X} := \big\{(\boldsymbol{x}_I,\boldsymbol{x}_C) \in \mathbb{Z}^{n_1}_{\geq} \times \mathbb{R}^{n_2}_{\geq} : A_1\boldsymbol{x}_I + A_2\boldsymbol{x}_C \leq \boldsymbol{b}\big\}$ represents the *feasible set in the decision space*, $\mathbb{Z}^{n_1}_{\geq} := \{\boldsymbol{s} \in \mathbb{Z}^{n_1} : \boldsymbol{s} \geq \boldsymbol{0}\}$, $\mathbb{R}^{n_2}_{\geq} := \{\boldsymbol{s} \in \mathbb{R}^{n_2} : \boldsymbol{s} \geq \boldsymbol{0}\}$, $A_1 \in \mathbb{R}^{m\times n_1}$, $A_2 \in \mathbb{R}^{m\times n_2}$, and $\boldsymbol{b} \in \mathbb{R}^m$. It is assumed that $\mathcal{X}$ is *bounded* and $z(\boldsymbol{x}_I,\boldsymbol{x}_C) = \boldsymbol{c}_I^\intercal\boldsymbol{x}_I + \boldsymbol{c}_C^\intercal\boldsymbol{x}_C$ where $\boldsymbol{c}_I \in \mathbb{R}^{n_1}$ and $\boldsymbol{c}_C \in \mathbb{R}^{n_2}$ represents a linear objective function. The image $\mathcal{Y}$ of $\mathcal{X}$ represents the *feasible set in the objective/criterion space*, that is $\mathcal{Y} := \{\boldsymbol{o} \in \mathbb{R} : \boldsymbol{o} = z(\boldsymbol{x}_I,\boldsymbol{x}_C)$ for all $(\boldsymbol{x}_I,\boldsymbol{x}_C) \in \mathcal{X}\}$. Clearly, $\boldsymbol{x}_I$ is a vector representing the integer decision variables and $\boldsymbol{x}_C$ is a vector representing the continuous decision variables.

2.2 Multi-Objective Mixed Integer Linear Programming

We now extend a MILP formulation by considering multiple objectives. This new formulation is known as Multi-Objective Mixed Integer Linear Programming and it can be stated as follows:

$$\min_{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}} \{z_1(\boldsymbol{x}_I,\boldsymbol{x}_C), z_2(\boldsymbol{x}_I,\boldsymbol{x}_C), \ldots, z_p(\boldsymbol{x}_I,\boldsymbol{x}_C)\}, \tag{2.2}$$

where $\mathcal{X}$ is *bounded* and $z_i(\boldsymbol{x}_I,\boldsymbol{x}_C) = \boldsymbol{c}_{i,I}^{\intercal}\boldsymbol{x}_I + \boldsymbol{c}_{i,C}^{\intercal}\boldsymbol{x}_C$ where $\boldsymbol{c}_{i,I} \in \mathbb{R}^{n_1}$ and $\boldsymbol{c}_{i,C} \in \mathbb{R}^{n_2}$ for $i = 1,2,\ldots,p$ represents a vector of p linear objective functions. The image $\mathcal{Y}$ of $\mathcal{X}$ under vector-valued function $\boldsymbol{z} := (z_1, z_2, \ldots, z_p)^{\intercal}$ represents the *feasible set in the objective/criterion space*, that is $\mathcal{Y} := \{\boldsymbol{o} \in \mathbb{R}^p : \boldsymbol{o} = \boldsymbol{z}(\boldsymbol{x}_I,\boldsymbol{x}_C) \text{ for all } (\boldsymbol{x}_I,\boldsymbol{x}_C) \in \mathcal{X}\}$.

Definition 1. A feasible solution $(\boldsymbol{x}_I,\boldsymbol{x}_C) \in \mathcal{X}$ is called *efficient*, if there is no other $(\boldsymbol{x}'_I,\boldsymbol{x}'_C) \in \mathcal{X}$ such that $z_i(\boldsymbol{x}'_I,\boldsymbol{x}'_C) < z_i(\boldsymbol{x}_I,\boldsymbol{x}_C)$ for at least one i where $i = 1,2,\ldots,p$, and $z_j(\boldsymbol{x}'_I,\boldsymbol{x}'_C) \leq z_j(\boldsymbol{x}_I,\boldsymbol{x}_C)$ for all j where $j = 1,2,\ldots,p$ such that $j \neq i$. If $(\boldsymbol{x}_I,\boldsymbol{x}_C)$ is efficient, then $\boldsymbol{z}(\boldsymbol{x}_I,\boldsymbol{x}_C)$ is called a *nondominated point*. The set of all efficient solutions is denoted by $\mathcal{X}_E$. The set of all nondominated points $\boldsymbol{z}(\boldsymbol{x}_I,\boldsymbol{x}_C)$ for $(\boldsymbol{x}_I,\boldsymbol{x}_C) \in \mathcal{X}_E$ is denoted by $\mathcal{Y}_N$ and referred to as the *nondominated frontier*.

Definition 2. If there exists a vector $(\lambda_1, \lambda_2, \ldots, \lambda_p)^{\intercal} \in \mathbb{R}^p_{>} := \{\boldsymbol{s} \in \mathbb{R}^p : \boldsymbol{s} > \boldsymbol{0}\}$ such that $(\boldsymbol{x}^*_I,\boldsymbol{x}^*_C) \in \arg\min_{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}} \lambda_1 z_1(\boldsymbol{x}_I,\boldsymbol{x}_C) + \lambda_2 z_2(\boldsymbol{x}_I,\boldsymbol{x}_C) + \cdots + \lambda_p z_p(\boldsymbol{x}_I,\boldsymbol{x}_C)$, then $(\boldsymbol{x}^*_I,\boldsymbol{x}^*_C)$ is called a *supported efficient solution* and $\boldsymbol{z}(\boldsymbol{x}^*_I,\boldsymbol{x}^*_C)$ is called a *supported nondominated point*.

Definition 3. Let $\mathcal{Y}^e$ be the set of extreme points of the convex hull of $\mathcal{Y}$, that is the smallest convex set containing the set $\mathcal{Y}$. A point $\boldsymbol{z}(\boldsymbol{x}_I,\boldsymbol{x}_C) \in \mathcal{Y}$ is called an *extreme supported nondominated point*, if $\boldsymbol{z}(\boldsymbol{x}_I,\boldsymbol{x}_C)$ is a supported nondominated point and $\boldsymbol{z}(\boldsymbol{x}_I,\boldsymbol{x}_C) \in \mathcal{Y}^e$.

2.3 Single-Objective Binary Quadratically Constrained Linear Programming

A single-objective binary quadratically constrained linear programming can be stated as follows:

$$\min_{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}} \; z(\boldsymbol{x}_I,\boldsymbol{x}_C), \tag{2.3}$$

where $\mathcal{X} := \big\{(\boldsymbol{x}_I,\boldsymbol{x}_C) \in \mathbb{Z}^{n_1}_{\geq} \times \mathbb{R}^{n_2}_{\geq} : (\boldsymbol{x}_I \frown \boldsymbol{x})^{\intercal} Q_i (\boldsymbol{x}_I \frown \boldsymbol{x}) + \boldsymbol{a}_{i1}^{\intercal}\boldsymbol{x}_I + \boldsymbol{a}_{i2}^{\intercal}\boldsymbol{x}_C \leq b_i$ for $i = 1,2,\ldots,m\big\}$ represents the *feasible set in the decision space*, $\mathbb{Z}^{n_1}_{\geq} := \{\boldsymbol{s} \in \mathbb{Z}^{n_1} : \boldsymbol{s} \geq \boldsymbol{0}\}$, $\mathbb{R}^{n_2}_{\geq} := \{\boldsymbol{s} \in \mathbb{R}^{n_2} : \boldsymbol{s} \geq \boldsymbol{0}\}$, $Q_i \in \mathbb{R}^{m\times(n_1+n_2)}$, $\boldsymbol{a}_{i1} \in \mathbb{R}^{n_1}$, $\boldsymbol{a}_{i2} \in \mathbb{R}^{n_2}$, and $b_i \in \mathbb{R}$ for $i = 1,2,\ldots,m$. Note that $(\boldsymbol{x}_I \frown \boldsymbol{x})$ is equivalent to the concatenation of the vectors $\boldsymbol{x}_I$ and $\boldsymbol{x}_C$ and m is the number of constraints.

This formulation is the basis for the Second Order Cone Programming (SOCP) formulation used in one of our contributions that considers a transformation of the Nash bargaining problem. An optimal solution for any SOCP formulation can be found using the most popular commercial solvers such as CPLEX, Gurobi annd Xpress.

Chapter 3: Optimization Over the Efficient Set of Bi-Objective Mixed Integer Linear Programs

In this chapter, we present the first two main contributions of this thesis. We first present some relevant literature and preliminaries for the problem of optimization over the efficient set of BOMILPs. Afterward, we introduce the first exact criterion space search algorithm for optimizing a linear function over the set of efficient solutions for BOMILPs. Finally, we develop a comprehensive open-source user-friendly package for BOMILPs. The copyright permissions for reuse previously published material in this chapter can be found in Appendix A1 and Appendix A2.

3.1 Introduction

Many real-world optimization problems involve multiple competing objectives, i.e., it is impossible to find a feasible solution that simultaneously optimizes all objectives. Consequently, it is not surprising that the focus of multi-objective optimization community has been primarily on developing effective techniques for generating some (if not all) *efficient* solutions, i.e., solutions in which it is impossible to improve the value of one objective without a deterioration in the value of at least one other objective, of multi-objective optimization problems. This is mainly because understanding the trade-offs between objectives can help decision makers select their preferred solutions. Interested readers may refer to [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1, 14, 15] for recent advances on exact solution approaches for multi-objective optimization problems.

Although understanding the trade-offs between objectives can be valuable, some researchers argue (see for instance Jorge [16]) that presenting too many efficient solutions can confuse a decision maker, and so may make selecting a preferred solution almost impossible. An approach that alleviates this issue is finding a preferred solution among the set of efficient solutions directly and is known as *optimizing over the efficient set*, which is a global optimization problem (Benson [17]). In this approach, the goal is to directly optimize a function over the set of efficient solutions (preferably without enumerating all of them).

It is worth mentioning that developing (effective) solution approaches for optimizing over the efficient set has two other benefits in addition to the fact that it can help decision makers find their perfected solutions:

1. It can help us solve one of the classical problems in the field of multi-objective optimization: computing the *nadir point*, i.e., the point in criterion space given by the worst value of each of the objective functions over the set of efficient solutions. Obviously, to compute each component of the nadir point, a special case of the problem of optimizing over the efficient set can be solved [18, 19, 20, 21].

2. It may help us solve certain single-objective optimization problems effectively. In other words, there are certain single-objective optimization problems that can be viewed as the problem of optimizing over the efficient set and can be solved faster in practice [22, 23, 24]. For example, $\max_{x\in\mathcal{X}} \prod_{i=1}^{p} z_i(x)$ when $z_i(x) \geq 0$ for all $x \in \mathcal{X}$ and $i = 1, \ldots, p$ is equivalent to the problem of maximizing $\prod_{i=1}^{p} z_i(x)$ over the set of efficient solutions of $\max_{x\in\mathcal{X}}\{z_1(x), \ldots, z_p(x)\}$.

It is worth mentioning there exist many studies about solving the problem of optimizing a linear function over the efficient set of multi-objective linear programs, see for instance Benson [17, 25, 26, 27], Dauer [28], Ecker and Song [29], Dayın [30] and Yamamoto [31]. Moreover, there are a few studies on optimizing a linear function over the set of efficient solutions of multi-objective pure integer linear programs, see for instance Abbas and Chaabane [32], Jorge [16], Chabaane and Pirlot [33], Chaabane et al. [34] and Boland et al. [18].

However, no exact algorithm is known for optimizing a linear function over the set of efficient solutions of multi-objective mixed integer linear programs (with an arbitrary number of objective functions). This is partly because, to this date, no exact algorithm has been developed for finding a full representation of *the nondominated frontier*, i.e., the set of points in criterion space corresponding to the set of efficient solutions, of multi-objective mixed integer linear programs with an arbitrary number of objective functions. If such an algorithm exists then they can be possibly modified for optimizing a linear function over the set of efficient solutions. However, fortunately, since a few algorithms have been recently developed for finding a full representation of the nondominated frontier of bi-objective mixed integer linear programs (see for instance Boland et al. [1], Fattahi and Turkay [35], Soylu and Yıldız [12], and Vincent et al. [36]), generating an exact algorithm for optimizing a linear function over the efficient set of bi-objective mixed integer linear programs should now be possible.

In light of the above, the main contribution of our research is to develop the first algorithm for optimizing a linear function over the set of efficient solutions of bi-objective mixed integer linear programs. Our algorithm employs the main components of the triangle splitting method which is one of the fastest exact *criterion space search algorithms*, i.e., methods that search in the space of objective function values, for computing a full representation of the nondominated frontier of a bi-objective mixed integer linear programs [1]. By conducting an extensive computational study, we demonstrate the efficacy of the proposed algorithm. The new algorithm has the following desirable characteristics:

- It is both easy to understand and easy to implement.
- It maintains a global lower bound and a global upper bound for the linear function at any point in time. So the algorithm is inherently good for quickly generating a provably high-quality approximate solution.
- The algorithm has minimal requirements in terms of information storage.

- The algorithm relies on solving single-objective mixed integer linear programs, and so it automatically benefits from any advances on single-objective mixed integer linear programming solvers.

The rest of this chapter is organized as follows. In Section 3.2, we explain the main concepts in bi-objective mixed integer linear programming and also provide a high-level description of the triangle splitting method and some necessary mathematical operations. We also describe the problem of optimizing over the efficient set and sketch the basis of our proposed algorithm. In Section 3.3, we present our proposed algorithm in detail, show an example of how the algorithm works, and evaluate its performance with a computational experiment. In Section 3.4, we present OOESAllgorithm.jl, an implementation of our algorithm in a comprehensive open-source user-friendly julia package. Finally, in Section 3.5, we provide some concluding remarks.

3.2 Preliminaries

In this section, we provide some preliminaries that are essential for explaining the contents of other sections.

3.2.1 Bi-Objective Mixed Integer Linear Programming

A *Bi-Objective Mixed Integer Linear Program* (BOMILP) can be stated as follows:

$$\min_{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}} \{z_1(\boldsymbol{x}_I,\boldsymbol{x}_C), z_2(\boldsymbol{x}_I,\boldsymbol{x}_C)\}, \tag{3.1}$$

where $\mathcal{X} := \big\{(\boldsymbol{x}_I,\boldsymbol{x}_C) \in \mathbb{Z}^{n_1}_{\geq} \times \mathbb{R}^{n_2}_{\geq} : A_1\boldsymbol{x}_I + A_2\boldsymbol{x}_C \leq \boldsymbol{b}\big\}$ represents the *feasible set in the decision space*, $\mathbb{Z}^{n_1}_{\geq} := \{\boldsymbol{s} \in \mathbb{Z}^{n_1} : \boldsymbol{s} \geq \boldsymbol{0}\}$, $\mathbb{R}^{n_2}_{\geq} := \{\boldsymbol{s} \in \mathbb{R}^{n_2} : \boldsymbol{s} \geq \boldsymbol{0}\}$, $A_1 \in \mathbb{R}^{m\times n_1}$, $A_2 \in \mathbb{R}^{m\times n_2}$, and $\boldsymbol{b} \in \mathbb{R}^m$. It is assumed that $\mathcal{X}$ is *bounded* and $z_i(\boldsymbol{x}_I,\boldsymbol{x}_C) = \boldsymbol{c}_{i,I}^{\intercal}\boldsymbol{x}_I + \boldsymbol{c}_{i,C}^{\intercal}\boldsymbol{x}_C$ where $\boldsymbol{c}_{i,I} \in \mathbb{R}^{n_1}$ and $\boldsymbol{c}_{i,C} \in \mathbb{R}^{n_2}$ for $i = 1, 2$ represents a linear objective function. The image $\mathcal{Y}$ of $\mathcal{X}$ under vector-valued function $\boldsymbol{z} := (z_1, z_2)^\intercal$ represents the *feasible set in the objective/criterion space*, that is $\mathcal{Y} := \{\boldsymbol{o} \in \mathbb{R}^2 : \boldsymbol{o} = \boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \text{ for all } (\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}\}$.

We now adapt some definitions from Section 2.2 to consider the special case of a MOMILP with 2 objectives functions.

Definition 4. A feasible solution $(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}$ is called *efficient* or, if there is no other $(\boldsymbol{x}'_I, \boldsymbol{x}'_C) \in \mathcal{X}$ such that $z_1(\boldsymbol{x}'_I, \boldsymbol{x}'_C) \leq z_1(\boldsymbol{x}_I, \boldsymbol{x}_C)$ and $z_2(\boldsymbol{x}'_I, \boldsymbol{x}'_C) < z_2(\boldsymbol{x}_I, \boldsymbol{x}_C)$ or $z_1(\boldsymbol{x}'_I, \boldsymbol{x}'_C) < z_1(\boldsymbol{x}_I, \boldsymbol{x}_C)$ and $z_2(\boldsymbol{x}'_I, \boldsymbol{x}'_C) \leq z_2(\boldsymbol{x}_I, \boldsymbol{x}_C)$. If $(\boldsymbol{x}_I, \boldsymbol{x}_C)$ is efficient, then $\boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C)$ is called a *nondominated point*. The set of all efficient solutions is denoted by $\mathcal{X}_E$. The set of all nondominated points $\boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C)$ for $(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}_E$ is denoted by $\mathcal{Y}_N$ and referred to as the *nondominated frontier*.

Definition 5. If there exists a vector $(\lambda_1, \lambda_2)^\intercal \in \mathbb{R}^2_> := \{\boldsymbol{s} \in \mathbb{R}^2 : \boldsymbol{s} > \boldsymbol{0}\}$ such that $(\boldsymbol{x}^*_I, \boldsymbol{x}^*_C) \in \arg\min_{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}} \lambda_1 z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) + \lambda_2 z_2(\boldsymbol{x}_I, \boldsymbol{x}_C)$, then $(\boldsymbol{x}^*_I, \boldsymbol{x}^*_C)$ is called a *supported efficient solution* and $\boldsymbol{z}(\boldsymbol{x}^*_I, \boldsymbol{x}^*_C)$ is called a *supported nondominated point*.

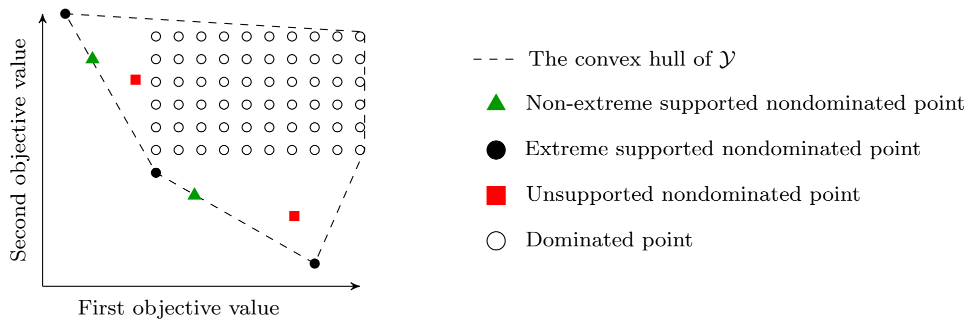


Figure 3.1 – An illustration of different types of (feasible) points in the criterion space

In summary, based on Definition 4, the elements of $\mathcal{Y}$ can be divided into two groups including dominated and nondominated points. Furthermore, based on Definitions 3 and 5, the nondominated points can be divided into unsupported nondominated points, non-extreme supported nondominated points and extreme supported nondominated points. Overall, bi-objective optimization problems are concerned with finding an exact representation of the elements of $\mathcal{Y}_N$, i.e., all nondominated points including supported and unsupported nondominated points. An illustration of the set $\mathcal{Y}$ when $n_2 = 0$, i.e., there is no continuous variable, and its corresponding categories are shown in Figure 3.1.

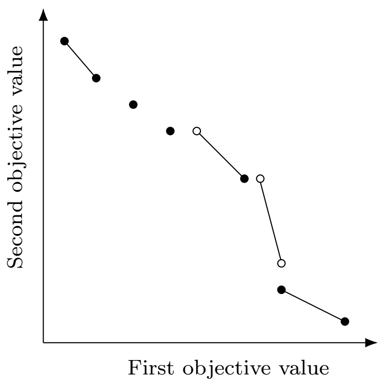


Figure 3.2 – An example of the nondominated frontier of a BOMILP

Note that if $n_1 > 0$ and $n_2 > 0$, the nondominated frontier of a BOMILP can be more complicated since some continuous segments may appear in the nondominated frontier. A typical example of a nondominated frontier of a BOMILP is shown in Figure 3.2. Observe that isolated points as well as closed, half-open, and open segments may exist in the nondominated frontier. So, although we assumed that $\mathcal{X}$ is bounded, we observe that the nondominated frontier of a BOMILP may still contain an infinite number of nondominated points (because of the existence of continuous segments) and so computing a (finite) exact representation of the nondominated frontier is quite challenging. So, that is one of the main reasons that only very few exact algorithms are developed for BOMILPs.

3.2.2 The Triangle Splitting Method

One of the effective (criterion space search) algorithms for BOMILPs is the Triangle Splitting Method (TSM) [1]. Since this algorithm provides the basis for the development of our algorithm, we next explain a high-level description of TSM.

TSM maintains a list of rectangles and right-angled triangles (in the criterion space) that need to be explored. At the beginning, this list is empty. So, TSM first computes the endpoints of the nondominated frontier. These two points are then used to define the first rectangle containing all the "not yet found" nondominated points, as shown in Figure 3.3a. The algorithm explores a rectangle by finding the locally extreme supported nondominated points within the rectangle. Now it can be shown that by finding these points, the rectangle

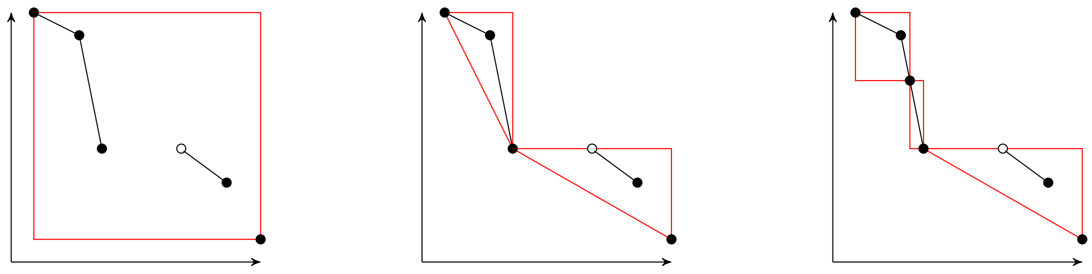

(a) Finding endpoints (b) Generated triangles (c) Generated rectangles

Figure 3.3 – Progression of TSM in terms of the discovery of nondominated points

can be split into a set of right-angled triangles containing all "not yet found" nondominated points, as shown in Figure 3.3b. The algorithm explores a triangle by first checking whether its hypotenuse is part of the nondominated frontier. If that is the case, then the triangle is removed from the list, otherwise it is split into at most two other rectangles. This operation is further illustrated in Figure 3.3c, where the hypotenuse of the top triangle in Figure 3.3b is not part of the nondominated frontier and so it is split into two new rectangles. The algorithm repeats these procedures until finding a full representation of the nondominated frontier, i.e., the list of rectangles and triangles becomes empty.

In light of the above, we now explain the notation and operations that are used in TSM and are necessary for the development of our algorithm. Let $\boldsymbol{z}^1 = (z_1^1, z_2^1)$ and $\boldsymbol{z}^2 = (z_1^2, z_2^2)$ be two points in the criterion space with $z_1^1 \leq z_1^2$ and $z_2^1 \geq z_2^2$. We denote by $R(\boldsymbol{z}^1, \boldsymbol{z}^2)$ the rectangle in the criterion space defined by the points $\boldsymbol{z}^1$ and $\boldsymbol{z}^2$. Furthermore, we denote by $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ the right-angled triangle in the criterion space defined by the points $\boldsymbol{z}^1$, (z_1^2, z_2^1), and $\boldsymbol{z}^2$. Finally, we denote by $H(\boldsymbol{z}^1, \boldsymbol{z}^2)$ the line segment in the criterion space defined by the points $\boldsymbol{z}^1$ and $\boldsymbol{z}^2$, i.e., the hypotenuse of triangle $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$.

3.2.2.1 Lexicographic Operation

The top endpoint, denoted by $\boldsymbol{z}^T$, of the nondominated frontier can be found by solving two (single-objective) Mixed Integer Linear Programs (MILPs) in sequence, as follows:

$$z_1^T = \min_{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}} z_1(\boldsymbol{x}_I,\boldsymbol{x}_C),$$

(if feasible) followed by

$$z_2^T = \min_{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}} \{z_2(\boldsymbol{x}_I,\boldsymbol{x}_C) : z_1(\boldsymbol{x}_I,\boldsymbol{x}_C) \leq z_1^T\}.$$

As this is an operation will be called frequently in our algorithm, we introduce the following notation to represent the process:

$$\boldsymbol{z}^T = \operatorname*{lex\,min}_{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}} \{z_1(\boldsymbol{x}_I,\boldsymbol{x}_C), z_2(\boldsymbol{x}_I,\boldsymbol{x}_C)\},$$

similarly, the bottom endpoint, denoted by $\boldsymbol{z}^B$, of the nondominated frontier can be found by solving:

$$\boldsymbol{z}^B = \operatorname*{lex\,min}_{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}} \{z_2(\boldsymbol{x}_I,\boldsymbol{x}_C), z_1(\boldsymbol{x}_I,\boldsymbol{x}_C)\}.$$

3.2.2.2 Weighted Sum Method Operation

Another frequently used operation is the weighted sum method ([37]). We denote this operation by Weighted-Sum-Method($\boldsymbol{z}^1, \boldsymbol{z}^2$) where $\boldsymbol{z}^1, \boldsymbol{z}^2 \in \mathcal{Y}_N$. This operation uses the following optimization problem to find all locally extreme supported nondominated points of a given rectangle $R(\boldsymbol{z}^1, \boldsymbol{z}^2)$:

$$\begin{aligned} \boldsymbol{z}^* = \min_{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}} \quad & \lambda_1 z_1(\boldsymbol{x}_I,\boldsymbol{x}_C) + \lambda_2 z_2(\boldsymbol{x}_I,\boldsymbol{x}_C) \\ \text{subject to} \quad & \boldsymbol{z}(\boldsymbol{x}_I,\boldsymbol{x}_C) \in R(\boldsymbol{z}^1, \boldsymbol{z}^2), \end{aligned}$$

where $\lambda_1, \lambda_2 > 0$. We note that only the value of λ_1 and λ_2 will be iteratively updated during the course of the weighted sum method (and not the set of constraints). In other words, in each iteration, for a given pair of nondominated points $(\boldsymbol{z}', \boldsymbol{z}'')$ with $z'_1 < z''_1$ and $z'_2 > z''_2$, we set $\lambda_1 = z'_2 - z''_2$ and $\lambda_2 = z''_1 - z'_1$. The first pair of points that will be used is $(\boldsymbol{z}^1, \boldsymbol{z}^2)$. It is not hard to see that using this pair, the value of objective function is parallel to the line that connects the points $\boldsymbol{z}^1$ and $\boldsymbol{z}^2$ in the criterion space as shown in Figure 3.4. We note that by Definition 5, the corresponding optimization problem always returns a new locally supported nondominated point $\boldsymbol{z}^*$. However, this point can be an extreme point only if $\lambda_1 z^*_1 + \lambda_2 z^*_2 < \lambda_1 z^1_1 + \lambda_2 z^1_2$. So, if this condition holds, the process will be repeated recursively for pairs $(\boldsymbol{z}^1, \boldsymbol{z}^*)$ and $(\boldsymbol{z}^*, \boldsymbol{z}^2)$. Overall, at the end of the weighted sum method, the list of points that we have discovered contains all locally extreme supported nondominated points as well as (possibly) some non-extreme supported nondominated points.

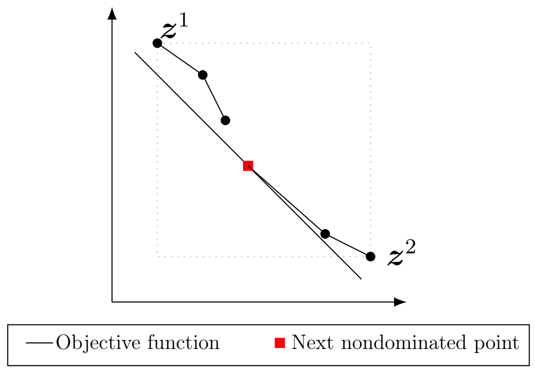


Figure 3.4 – Weighted sum optimization problem

3.2.2.3 Line Detector Operation

This operation is a (single-objective) MILP that determines whether some or all points on the hypotenuse of a given triangle $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ are nondominated points based on the following theorem, and it is denoted by Line-Detector$(\boldsymbol{z}^1, \boldsymbol{z}^2)$.

Theorem 1. Let $\boldsymbol{z}^1$ and $\boldsymbol{z}^2$ be two nondominated points in the criterion space. If $(H(\boldsymbol{z}^1, \boldsymbol{z}^2) - \mathbb{R}^2_{>}) \cap \mathcal{Y}_N = \emptyset$ and there exists an $\boldsymbol{x}_I \in \mathcal{X}_I$ and $\boldsymbol{x}^1_C$ and $\boldsymbol{x}^2_C \in \mathbb{R}^m$ such that

$(\boldsymbol{x}_I, \boldsymbol{x}_C^1), (\boldsymbol{x}_I, \boldsymbol{x}_C^2) \in \mathcal{X}$, $z_1(\boldsymbol{x}_I, \boldsymbol{x}_C^1) \leq z_1^1$, $z_2(\boldsymbol{x}_I, \boldsymbol{x}_C^1) \leq z_2^1$, $z_1(\boldsymbol{x}_I, \boldsymbol{x}_C^2) \leq z_1^2$, and $z_2(\boldsymbol{x}_I, \boldsymbol{x}_C^2) \leq z_2^2$, then $H(\boldsymbol{z}^1, \boldsymbol{z}^2) \subseteq \mathcal{Y}_N$. Boland et al. [1]

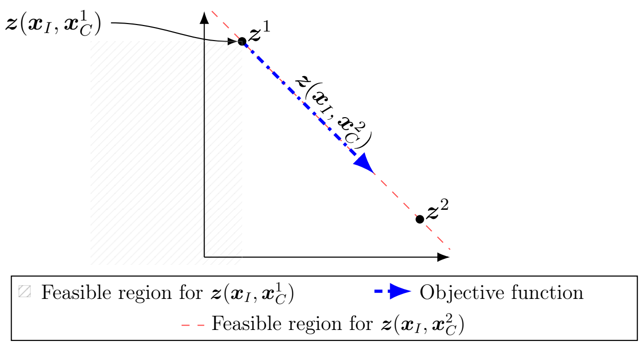
Specifically, suppose that for a given triangle $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ generated during the course of TSM, we simultaneously find two feasible solutions $(\boldsymbol{x}_I, \boldsymbol{x}_C^1), (\boldsymbol{x}_I, \boldsymbol{x}_C^2) \in \mathcal{X}$ such that the following three conditions hold:

- If there is a difference between solutions, it is only because of the values of continuous decision variables.
- For the first solution, we must have that $\boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C^1) = \boldsymbol{z}^1$.
- For the second solution, we must have that $\boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C^2) \in H(\boldsymbol{z}^1, \boldsymbol{z}^2)$.

If these conditions hold at the same time then the line segment between the point $\boldsymbol{z}^1$ and $\boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C^2)$ is part of the nondominated frontier. Obviously, such a line segment is a subset of $H(\boldsymbol{z}^1, \boldsymbol{z}^2)$, so the goal would be to maximize the length of the line segment to hopefully cover the entire hypotenuse. This can be achieved by solving the following MILP:

$$\begin{aligned}
\max\ & z_1(\boldsymbol{x}_I, \boldsymbol{x}_C^2) \\
\text{subject to}\ & z_1(\boldsymbol{x}_I, \boldsymbol{x}_C^1) \leq z_1^1 \\
& z_2(\boldsymbol{x}_I, \boldsymbol{x}_C^1) \leq z_2^1 \\
& \lambda_1 z_1(\boldsymbol{x}_I, \boldsymbol{x}_C^2) + \lambda_2 z_2(\boldsymbol{x}_I, \boldsymbol{x}_C^2) = \lambda_1 z_1^1 + \lambda_2 z_2^1 \\
& (\boldsymbol{x}_I, \boldsymbol{x}_C^1) \in \mathcal{X}, (\boldsymbol{x}_I, \boldsymbol{x}_C^2) \in \mathcal{X}
\end{aligned}$$

where $\lambda_1 = z_2^1 - z_2^2$ and $\lambda_2 = z_1^2 - z_1^1$. Note that the first and second constraints can be written in the form of equality as well but since $\boldsymbol{z}^1$ is a nondominated point it is computationally better to use inequalities. An illustration of this optimization problem can be found in Figure 3.5.



$z(x_I, x_C^1)$
z^1
$z(x_I, x_C^2)$
z^2
▨ Feasible region for $z(x_I, x_C^1)$
Objective function
Feasible region for $z(x_I, x_C^2)$

Figure 3.5 – Line detector operation

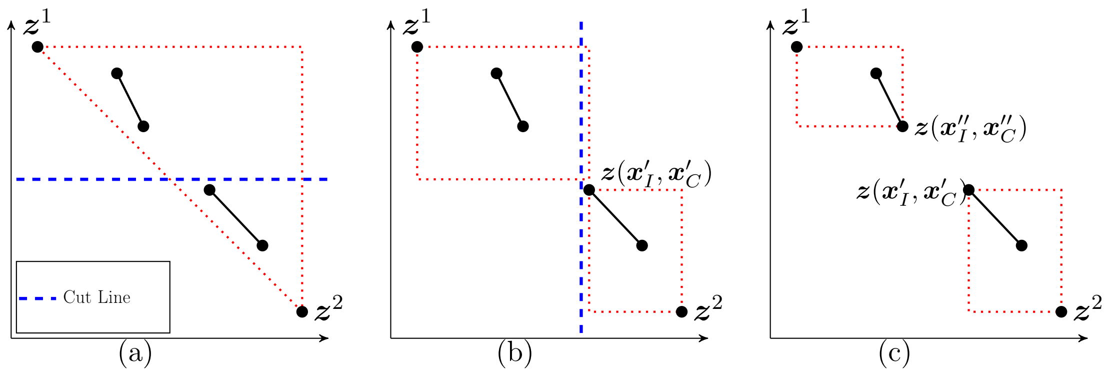


Figure 3.6 – Horizontal splitting of triangle $\mathbf{T}(z^1, z^2)$

3.2.2.4 Triangle Split Operation

This operation splits a given triangle $T(z^1, z^2)$. We denote this operation by Split-Triangle($z^1, z^2, horizontal$) or Split-Triangle($z^1, z^2, vertical$) when we split a triangle first horizontally (and then possibly vertically) or first vertically (and then possibly horizontally), respectively. We now explain Split-Triangle($z^1, z^2, horizontal$) graphically as shown in Figure 3.6.

In this operation, as shown in Figure 3.6a, we first cut the triangle into two parts horizontally and then explore the bottom part of the triangle to compute the top (local) endpoint of the nondominated frontier in the search region. This can be done using the following lexicographic operation:

$$(x'_I, x'_C) \in \arg \operatorname{lex} \min_{(x_I, x_C) \in \mathcal{X}} \{z_1(x_I, x_C), z_2(x_I, x_C) : z_2(x_I, x_C) \le \frac{z_2^1 + z_2^2}{2}, z(x_I, x_C) \in T(z^1, z^2)\},$$

and next, as shown in Figure 3.6b, we cut the triangle vertically based on the position of $\boldsymbol{z}(\boldsymbol{x}'_I, \boldsymbol{x}'_C)$ and explore the left part of the triangle to compute the bottom (local) endpoint of the nondominated frontier in that search region. This can be done using the following lexicographic operation:

$$(\boldsymbol{x}''_I, \boldsymbol{x}''_C) \in \underset{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}}{\arg \operatorname{lex}\min} \ \{z_2(\boldsymbol{x}_I, \boldsymbol{x}_C), z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) : z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) < z_1(\boldsymbol{x}'_I, \boldsymbol{x}'_C),$$
$$\boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in T(\boldsymbol{z}^1, \boldsymbol{z}^2)\}.$$

Finally, as shown in Figure 3.6c, after computing these two new nondominated points, the triangle can be changed to (at most) two new rectangles $R\big(\boldsymbol{z}^1, \boldsymbol{z}(\boldsymbol{x}''_I, \boldsymbol{x}''_C)\big)$ and $R\big(\boldsymbol{z}(\boldsymbol{x}'_I, \boldsymbol{x}'_C), \boldsymbol{z}^2\big)$. Obviously, there is no need to consider $R\big(\boldsymbol{z}^1, \boldsymbol{z}(\boldsymbol{x}''_I, \boldsymbol{x}''_C)\big)$ for further investigation if $\boldsymbol{z}^1 = \boldsymbol{z}(\boldsymbol{x}''_I, \boldsymbol{x}''_C)$. Similarly, there is no need to consider $R\big(\boldsymbol{z}(\boldsymbol{x}'_I, \boldsymbol{x}'_C), \boldsymbol{z}^2\big)$ for further investigation if $\boldsymbol{z}(\boldsymbol{x}'_I, \boldsymbol{x}'_C) = \boldsymbol{z}^2$. As an aside, to avoid spending the valuable computational time on (potential) redundant calculations, we set $(\boldsymbol{x}''_I, \boldsymbol{x}''_C) = (\boldsymbol{x}'_I, \boldsymbol{x}'_C)$ and skip solving the second lexicographic operation if the image of $(\boldsymbol{x}'_I, \boldsymbol{x}'_C)$ in the criterion space is on the horizontal cut line, i.e., $z_2(\boldsymbol{x}'_I, \boldsymbol{x}'_C) = \frac{z_2^1 + z_2^2}{2}$.

We note that Split-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2, \textit{vertical}$) can be defined similarly. In other words, we first need to compute:

$$(\boldsymbol{x}''_I, \boldsymbol{x}''_C) \in \underset{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}}{\arg \operatorname{lex}\min} \ \{z_2(\boldsymbol{x}_I, \boldsymbol{x}_C), z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) : z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq \frac{z_1^1 + z_1^2}{2}, \boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in T(\boldsymbol{z}^1, \boldsymbol{z}^2)\},$$

and then follow it by:

$$(\boldsymbol{x}'_I, \boldsymbol{x}'_C) \in \underset{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}}{\arg \operatorname{lex}\min} \ \{z_1(\boldsymbol{x}_I, \boldsymbol{x}_C), z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) : z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) < z_2(\boldsymbol{x}''_I, \boldsymbol{x}''_C),$$
$$\boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in T(\boldsymbol{z}^1, \boldsymbol{z}^2)\}.$$

It is worth mentioning that TSM (and our proposed algorithm in this study) alternates between Split-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2, \textit{horizontal}$) and Split-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2, \textit{vertical}$) to be able to discover both horizontal and vertical gaps in the nondominated frontier.

3.2.3 Optimization Over the Efficient Set of a BOMILP

The problem of optimizing a linear function over the set of efficient solutions of a BOMILP can be stated as follows:

$$\min_{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}_E} \quad f(\boldsymbol{x}_I,\boldsymbol{x}_C) \tag{3.2}$$

where $\mathcal{X}_E$ is the set of efficient solutions of Problem (3.1) and $f(\boldsymbol{x}_I,\boldsymbol{x}_C) = \boldsymbol{c}_{f,I}^{\intercal}\boldsymbol{x}_I + \boldsymbol{c}_{f,C}^{\intercal}\boldsymbol{x}_C$ where $\boldsymbol{c}_{f,I} \in \mathbb{R}^{n_1}$ and $\boldsymbol{c}_{f,C} \in \mathbb{R}^{n_2}$ represents a linear function. To ensure that the problem cannot be solved straightforwardly, we assume that $\mathcal{X} \neq \mathcal{X}_E$ and $f(\boldsymbol{x})$ is not a strictly positive linear combination of $z_1(\boldsymbol{x})$ and $z_2(\boldsymbol{x})$.

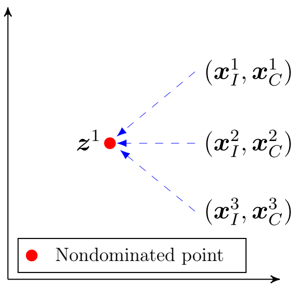

$(\boldsymbol{x}_I^1,\boldsymbol{x}_C^1)$
$\boldsymbol{z}^1$
$(\boldsymbol{x}_I^2,\boldsymbol{x}_C^2)$
$(\boldsymbol{x}_I^3,\boldsymbol{x}_C^3)$
● Nondominated point

Figure 3.7 – Image of some efficient solutions in the criterion space

3.3 A New Exact Algorithm for Optimizing a Linear Function Over the Efficient Set of BOMILPs

In this section, we explain in detail the key operations and the details of the proposed algorithm.

3.3.1 Key Operations

First, we provide some notation and operations that form the basis for the new algorithm. However, before that we make two important observations.

Observation 1. There may exist one or more efficient solutions with the same image in the criterion space. So, after discovering a nondominated point, we often need to find a solution among all feasible solutions corresponding to that point which has the minimum value for $f(\boldsymbol{x}_I, \boldsymbol{x}_C)$. For example, in Figure 3.7, it is observed that $(\boldsymbol{x}_I^1, \boldsymbol{x}_C^1)$, $(\boldsymbol{x}_I^2, \boldsymbol{x}_C^2)$ and $(\boldsymbol{x}_I^3, \boldsymbol{x}_C^3)$ return the same nondominated point. Now, if we have that $f(\boldsymbol{x}_I^1, \boldsymbol{x}_C^1) < f(\boldsymbol{x}_I^2, \boldsymbol{x}_C^2) < f(\boldsymbol{x}_I^3, \boldsymbol{x}_C^3)$ then $(\boldsymbol{x}_I^1, \boldsymbol{x}_C^1)$ provides the best upper (or primal) bound for the optimal value of Problem (3.2).

Observation 2. Let $f^l := \min_{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}} \ f(\boldsymbol{x}_I, \boldsymbol{x}_C)$ and $f^* = \min_{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}_E} \ f(\boldsymbol{x}_I, \boldsymbol{x}_C)$. We must have that $f^l \leq f^*$, i.e., f^l is a lower (or dual) bound.

Overall, we use the notation $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) \in \mathcal{X}$ for a local lower bound solution and $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \in \mathcal{X}_E$ for a local upper bound solution. We now describe a few necessary operations to compute $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$ and $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$.

3.3.1.1 Computing Local Upper Bound Solutions

During the course of the new algorithm, e.g., when splitting a triangle, we often discover a nondominated point $\boldsymbol{z}^1 \in \mathcal{Y}_N$. In that case, based on Observation 1, we solve the following optimization problem to obtain a local upper bound solution. We denote this operation by UB-Finder-Point($\boldsymbol{z}^1$):

$$(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \in \underset{x\in\mathcal{X}}{\arg\min} \ \{f(\boldsymbol{x}) : \boldsymbol{z}(\boldsymbol{x}) \leq \boldsymbol{z}^1\}.$$

When we are exploring a right-angeled triangle $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ during the course of the proposed algorithm, we may discover that the hypotenuse of the triangle is part of the nondominated frontier, i.e., $H(\boldsymbol{z}^1, \boldsymbol{z}^2) \in \mathcal{Y}_N$. So, in this case, based on Observation 1, we solve the following optimization problem to obtain a local upper bound solution. We denote this operation by UB-Finder-Line($\boldsymbol{z}^1, \boldsymbol{z}^2$):

$$(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \in \underset{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}}{\arg\min} \ \{f(\boldsymbol{x}_I, \boldsymbol{x}_C) : \boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in H(\boldsymbol{z}^1, \boldsymbol{z}^2)\}.$$

3.3.1.2 Computing Local Lower Bound Solutions

Before exploring a rectangle $R(\boldsymbol{z}^1, \boldsymbol{z}^2)$ during the course of the proposed algorithm, based on Observation 2, we solve the following optimization problem to obtain a local lower bound solution. We denote this operation by LB-Finder-Rectangle($\boldsymbol{z}^1, \boldsymbol{z}^2$):

$$(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) \in \underset{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}}{\arg\min} \ \{f(\boldsymbol{x}_I, \boldsymbol{x}_C) : \boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in R(\boldsymbol{z}^1, \boldsymbol{z}^2)\}.$$

Similarly, before exploring a triangle $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ during the course of the proposed algorithm, based on Observation 2, we solve the following optimization problem to obtain a local lower bound solution. We denote this operation by LB-Finder-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2$):

$$(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) \in \underset{x \in \mathcal{X}}{\arg\min} \ \{f(\boldsymbol{x}_I, \boldsymbol{x}_C) : \boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in T(\boldsymbol{z}^1, \boldsymbol{z}^2)\}.$$

3.3.1.3 Computing an Efficient Solution Based on a Local Lower Solution

After computing a local lower bound solution $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$ by using LB-Finder-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2$), we often try to compute an efficient solution in $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ where $\boldsymbol{z}^1, \boldsymbol{z}^2 \in \mathcal{Y}_N$, based on $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$. In order to do so, we solve the following optimization problem. We denote this operation by Find-NDP($\boldsymbol{z}(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$):

$$(\boldsymbol{x}_I', \boldsymbol{x}_C') \in \underset{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}}{\arg\min} \ \{z_1(\boldsymbol{x}) + z_2(\boldsymbol{x}) : \boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq \boldsymbol{z}(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)\}.$$

Note that showing that this operation returns an efficient solution within the search region defined by $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ where $\boldsymbol{z}^1, \boldsymbol{z}^2 \in \mathcal{Y}_N$ is straightforward, and so we have omitted the proof. Note too that computing $(\boldsymbol{x}_I', \boldsymbol{x}_C')$ is useful since we often use it as an alternative to split $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$. This implies that after computing $(\boldsymbol{x}_I', \boldsymbol{x}_C')$, we often immediately change $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ into two new rectangles, i.e., $R(\boldsymbol{z}^1, \boldsymbol{z}(\boldsymbol{x}_I', \boldsymbol{x}_C'))$ and $R(\boldsymbol{z}(\boldsymbol{x}_I', \boldsymbol{x}_C'), \boldsymbol{z}^2)$.

3.3.2 The Algorithm

The new algorithm maintains a priority queue of rectangles and triangles. Specifically, each element of the priority queue is denoted by $(\boldsymbol{z}^1, \boldsymbol{z}^2, shape, direction, LB)$ where 'shape' is either triangle or rectangle; '$\boldsymbol{z}^1$' and '$\boldsymbol{z}^2$' are the corner points of the corresponding rectangle or triangle; 'direction' is either horizontal or vertical and it indicates that if the shape is a triangle and we want to split it then how this process should be done; and 'LB' is the (local) lower bound corresponding to this element (obtained by using the techniques developed in subsection 3.3.1.2).

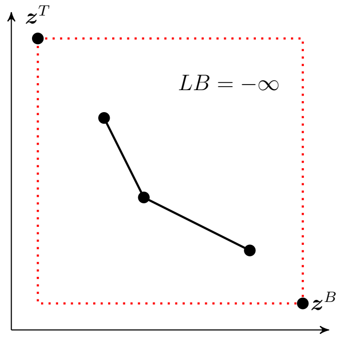


Figure 3.8 – Initial rectangle

At the first iteration of the algorithm, the priority queue is initialized with $(\boldsymbol{z}^T, \boldsymbol{z}^B,$ $rectangle, horizontal, -\infty)$. An illustration of the initial rectangle can be found in Figure 3.8. The algorithm also maintains a global upper bound and global lower bound for Problem (3.2) denoted by GUB and GLB, respectively. Let $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*) \in \mathcal{X}_E$ be the best efficient solution that has been found at any point during the course of the algorithm. Obviously, $GUB = f(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*)$. Note that we set $GUB = +\infty$ at the beginning of the algorithm. Also, the GLB is equal to the minimum value of LB in all elements of the priority queue. We assume that the priority queue is sorted in nondecreasing order of the value of LB at any point in time. Consequently, the local lower bound corresponding to the first element of the list is always GLB. It is worth mentioning that the algorithm terminates as soon as the priority queue is empty or the optimality gap is below a certain threshold. Note that if the priority queue becomes empty then the optimality gap is naturally zero. Next, we explain how the algorithm works in each iteration.

In each iteration, the algorithm pops out the first element of the priority queue and denote it by $(\boldsymbol{z}^1, \boldsymbol{z}^2, shape, direction, LB)$. Note that when an element is popped out from the priority queue then that element does not exist in the priority queue anymore. The algorithm first sets $GUB = LB$ and then computes both the relative and absolute optimality gaps. Let $\epsilon_1, \epsilon_2 > 0$ be small positive values that are defined by users. If $\frac{|GUB-GLB|}{|GUB|+\varepsilon_1} \leq \varepsilon_2$ or $GUB \leq GLB + \varepsilon_2$ then the algorithm terminates and reports $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*)$ as an optimal solution. Note that ε_1 is just introduced to modify the denominator of the relative gap for cases with $GUB = 0$. So, in this study, we simply assume that $\varepsilon_1 = 10^{-5}$.

So, in the remaining, we assume that $\frac{|GUB-GLB|}{|GUB|+\varepsilon_1} > \varepsilon_2$ and $GUB > GLB + \varepsilon_2$. In this case, the algorithm first checks whether $\boldsymbol{z}^1 = \boldsymbol{z}^2$. If that is the case then the algorithm calls UB-Finder-Point($\boldsymbol{z}^1$) to compute an upper bound solution $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$ and then the new iteration starts after updating GUB and $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*)$. Note that, from now on, updating GUB and $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*)$ means that if $GUB > f(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$ then we set $GUB = f(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$ and $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*) = (\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$.

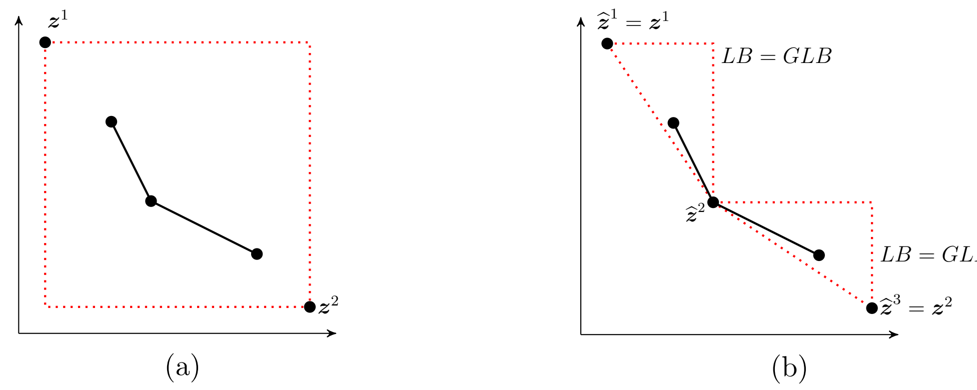

$\boldsymbol{z}^1$
$\boldsymbol{z}^2$
(a)
$\widehat{\boldsymbol{z}}^1 = \boldsymbol{z}^1$
$LB = GLB$
$\widehat{\boldsymbol{z}}^2$
$LB = GLB$
$\widehat{\boldsymbol{z}}^3 = \boldsymbol{z}^2$
(b)

Figure 3.9 – Changing rectangle $R(\boldsymbol{z}^1, \boldsymbol{z}^2)$ into smaller triangles

So, in the remaining, we assume that $\boldsymbol{z}^1 \neq \boldsymbol{z}^2$. Now, if the shape is rectangle then the algorithm simply applies the weighted sum method in the rectangle to find all locally extreme supported nondominated points. We denote the outcome of the weighted sum operation by

$\widehat{\boldsymbol{z}}^1, \widehat{\boldsymbol{z}}^2, \ldots, \widehat{\boldsymbol{z}}^k$ such that $\widehat{z}_1^v < \widehat{z}_1^{v+1}$ and $\widehat{z}_2^v > \widehat{z}_2^{v+1}$ for each $v = 1, \ldots, k-1$. After finding these points, the algorithm adds $k-1$ new elements to the priority queue that are all triangles. Specifically, it adds $(\widehat{\boldsymbol{z}}^v, \widehat{\boldsymbol{z}}^{v+1}, \mathit{triangle}, \mathit{direction}, GLB)$ to the priority queue for each $v = 1, \ldots, k-1$. Note that the direction and the local global lower bound of the new elements are as same the direction and the local global lower bound of the rectangle that is explored (we know that, by construction, $GLB = LB$). It is worth mentioning that the algorithm does not attempt to find better lower bounds for the new triangles simply because we have observed that this is computationally expensive (in practice). An illustration of generated triangles after exploring $R(\boldsymbol{z}^1, \boldsymbol{z}^2)$ can be found in Figure 3.9.

Now, if the shape is triangle, i.e., $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$, then the algorithm checks whether $H(\boldsymbol{z}^1, \boldsymbol{z}^2)$ is partially or entirely part of the nondominated frontier. In other words, the algorithm calls Line-Detector$(\boldsymbol{z}^1, \boldsymbol{z}^2)$ and denotes its corresponding optimal solution by $(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C)$. From the theory of the line detector operation (presented in subsection 3.2.2), we know that the line segment between $\boldsymbol{z}^1$ and $(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C)$ should be part of the nondominated frontier. So, now one of the two possible cases may arise:

- Case I: $\boldsymbol{z}^1 \neq \boldsymbol{z}(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C)$ (which is equivalent to $z_1(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C) > z_1^1$ based on the construction of the line detector operation)

- Case II: $\boldsymbol{z}^1 = \boldsymbol{z}(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C)$ (which is equivalent to $z_1(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C) = z_1^1$ based on the construction of the line detector operation)

If Case I arises then the algorithm calls UB-Finder-Line$\big(\boldsymbol{z}^1, \boldsymbol{z}(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C)\big)$ to compute an upper bound solution $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$ and then updates GUB and $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*)$. Finally, if the hypotenuse is not entirely part of the nondominated frontier, i.e., $z_1(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C) < z_1^2$, then the algorithm adds $(\boldsymbol{z}(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C), \boldsymbol{z}^2, \mathit{triangle}, \mathit{direction}, GLB)$ to the priority. Note that the new element is a smaller version of the initial triangle, i.e., $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$. An illustration of such a triangle can be found in Figure 3.10.

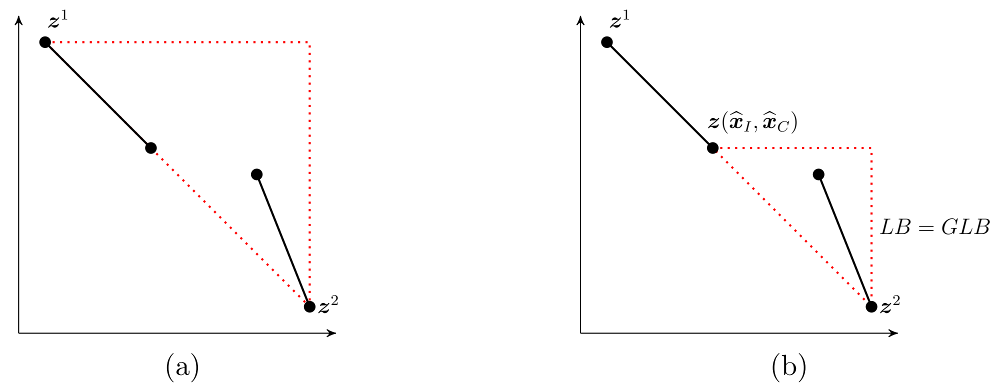


Figure 3.10 – Generating a smaller triangle after observing that the hypotenuse of is partially part of the nondominated frontier

If Case II arises then the algorithm first calls LB-Finder-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2$) to find a lower bound solution $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$ within the search region defined by the triangle. Obviously, if $f(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) \geq GUB$ then there is no need to explore the triangle and so the algorithm starts a new iteration. Otherwise, we need to split the triangle, i.e., $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$, into two new rectangles, denoted by $R(\boldsymbol{z}^1, \underline{\boldsymbol{z}})$ and $R(\bar{\boldsymbol{z}}, \boldsymbol{z}^2)$ where $\bar{\boldsymbol{z}}, \underline{\boldsymbol{z}} \in \mathcal{Y}_N$ and $\bar{\boldsymbol{z}}, \underline{\boldsymbol{z}} \in T(\boldsymbol{z}^1, \boldsymbol{z}^2)$. We also denoted by $\underline{L}$ and $\bar{L}$ a (local) lower bound corresponding to rectangles $R(\boldsymbol{z}^1, \underline{\boldsymbol{z}})$ and $R(\bar{\boldsymbol{z}}, \boldsymbol{z}^2)$ respectively. In order to compute $\bar{\boldsymbol{z}}, \underline{\boldsymbol{z}}, \underline{L}$ and $\bar{L}$, we have developed an operation denoted by Explore-Triangle$\big((\boldsymbol{x}_I^l, \boldsymbol{x}_C^l), \boldsymbol{z}^1, \boldsymbol{z}^2, direction\big)$. This operation, in addition to $\bar{\boldsymbol{z}}, \underline{\boldsymbol{z}}, \underline{L}$ and $\bar{L}$, returns a new 'direction' and a new upper bound solution $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$ as well. Later in this section, we explain this operation in detail.

In light of the above, after calling Explore-Triangle$\big((\boldsymbol{x}_I^l, \boldsymbol{x}_C^l), \boldsymbol{z}^1, \boldsymbol{z}^2, direction\big)$, the algorithm first updates GUB and $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*)$ using the new upper bound solution $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$. The algorithm then adds the element $(\boldsymbol{z}^1, \underline{\boldsymbol{z}}, rectangle, direction, \underline{L})$ to the priority queue if $\boldsymbol{z}^1 \neq \underline{\boldsymbol{z}}$ (since otherwise the shape is a single point and not a rectangle) and $\underline{L} < GUB$ (since otherwise such a rectangle cannot contain a better solution). Similarly, the algorithm adds $(\bar{\boldsymbol{z}}, \boldsymbol{z}^2, rectangle, direction, \bar{L})$ to the priority queue if $\boldsymbol{z}^2 \neq \bar{\boldsymbol{z}}$ and $\bar{L} < GUB$.

A detailed description of the proposed algorithm can be found in Algorithm 3.

Algorithm 1: The new algorithm

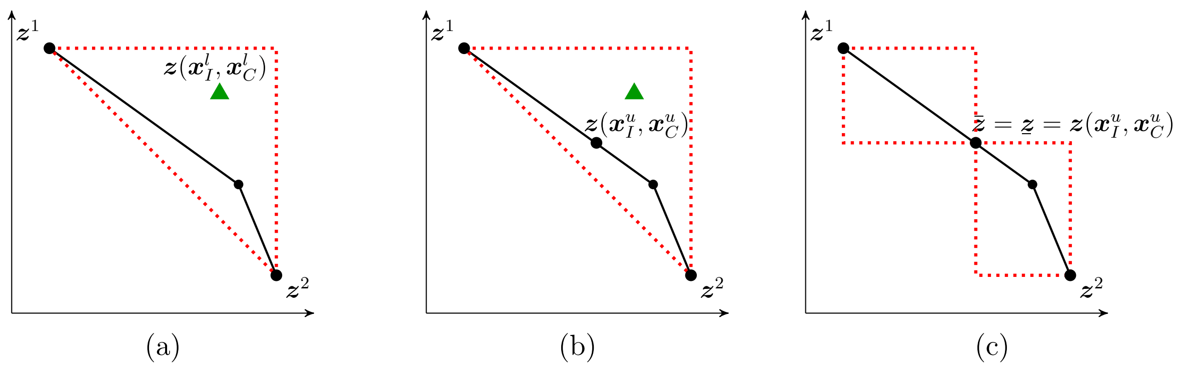
```
$GLB \leftarrow -\infty$; $GUB \leftarrow +\infty$; $PQ.create(P)$; $PQ.add(P, (\boldsymbol{z}^T, \boldsymbol{z}^B, rectangle, horizontal, -\infty))$
$Search\_Done \leftarrow False$
**while** $not\ PQ.empty(P)$ **and** $Search\_Done = False$ **do**
    $PQ.pop(P, (\boldsymbol{z}^1, \boldsymbol{z}^2, shape, direction, LB))$
    $GLB \leftarrow LB$
    **if** $\frac{|GUB-GLB|}{|GUB|+\varepsilon_1} \leq \varepsilon_2$ **or** $GUB \leq GLB + \varepsilon_2$ **then**
        $Search\_Done \leftarrow True$
    **else**
        **if** $\boldsymbol{z}^1 = \boldsymbol{z}^2$ **then**
            $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \leftarrow$ UB-Finder-Point$(\boldsymbol{z}^1)$
            **if** $f(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) < GUB$ **then**
                $GUB \leftarrow f(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$
                $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*) \leftarrow (\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$
        **else**
            **if** $shape = rectangle$ **then**
                $\{\widehat{\boldsymbol{z}}^1, \widehat{\boldsymbol{z}}^2, ..., \widehat{\boldsymbol{z}}^k\} \leftarrow$ Weighted-Sum-Method$(\boldsymbol{z}^1, \boldsymbol{z}^2)$
                $i \leftarrow 1$
                **while** $i < k$ **do**
                    $PQ.add(P, (\widehat{\boldsymbol{z}}^i, \widehat{\boldsymbol{z}}^{i+1}, triangle, direction, GLB)$
                    $i \leftarrow i + 1$
            **else**
                $(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C) \leftarrow$ Line-Detector$(\boldsymbol{z}^1, \boldsymbol{z}^2)$
                **if** $z_1(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C) > z_1^1$ **then**
                    $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \leftarrow$ UB-Finder-Line$\big(\boldsymbol{z}^1, \boldsymbol{z}(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C)\big)$
                    **if** $f(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) < GUB$ **then**
                        $GUB \leftarrow f(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$
                        $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*) \leftarrow (\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$
                    **if** $z_1(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C) < z_1^2$ **then**
                        $PQ.add(P, (\boldsymbol{z}(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C), \boldsymbol{z}^2, triangle, direction, GLB))$
                **else**
                    $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) \leftarrow$ LB-Finder-Triangle$(\boldsymbol{z}^1, \boldsymbol{z}^2)$
                    **if** $f(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) < GUB$ **then**
                        $\big((\boldsymbol{x}_I^u, \boldsymbol{x}_C^u), (\underline{\boldsymbol{z}}, \underline{L}), (\bar{\boldsymbol{z}}, \bar{L}), direction\big) \leftarrow$
                          Explore-Triangle $\big((\boldsymbol{x}_I^l, \boldsymbol{x}_C^l), \boldsymbol{z}^1, \boldsymbol{z}^2, direction\big)$
                        **if** $f(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) < GUB$ **then**
                            $GUB \leftarrow f(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$
                            $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*) \leftarrow (\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$
                        **if** $\underline{L} < GUB$ **and** $\boldsymbol{z}^1 \neq \underline{\boldsymbol{z}}$ **then**
                            $PQ.add(P, (\boldsymbol{z}^1, \underline{\boldsymbol{z}}, rectangle, direction, \underline{L}))$
                        **if** $\bar{L} < GUB$ **and** $\boldsymbol{z}^2 \neq \bar{\boldsymbol{z}}$ **then**
                            $PQ.add(P, (\bar{\boldsymbol{z}}, \boldsymbol{z}^2, rectangle, direction, \bar{L}))$
**return** $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*)$ and $f(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*)$
```

3.3.2.1 Operation Explore Triangle

We now explain the details of Explore-Triangle$\big((\boldsymbol{x}_I^l, \boldsymbol{x}_C^l), \boldsymbol{z}^1, \boldsymbol{z}^2, \mathit{direction}\big)$. Evidently, the inputs of this operation are $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$, $\boldsymbol{z}^1$, $\boldsymbol{z}^2$ and *direction*. The outputs of this operation are $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u), (\underline{\boldsymbol{z}}, \underline{L}), (\bar{\boldsymbol{z}}, \bar{L})$ and *direction*.

As discussed previously, the operation provides the necessary components for splitting the right-angeled triangle $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ into (at most) two new rectangles. This operation first checks whether the image of $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$ in the criterion space is (reasonably) far away from the orthogonal sides of the right-angeled triangle $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ (the reason for checking this will become clear in the upcoming paragraphs). In other words, it checks whether $z_1(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) < z_1^2 - (z_1^2 - z_1^1)\varepsilon_3$ and $z_2(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) < z_2^1 - (z_2^1 - z_2^2)\varepsilon_3$ where $\epsilon_3 \in (0, 1)$ is a user defined parameter. The default value of ϵ_3 is 0.15 in our implementation of the algorithm since this value performs the best for all our computational experiments. So, two possible cases may arise:

- Case A: $z_1(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) < z_1^2 - (z_1^2 - z_1^1)\varepsilon_3$ and $z_2(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) < z_2^1 - (z_2^1 - z_2^2)\varepsilon_3$.
- Case B: $z_1(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) \geq z_1^2 - (z_1^2 - z_1^1)\varepsilon_3$ or $z_2(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) \geq z_2^1 - (z_2^1 - z_2^2)\varepsilon_3$ (or both).


$\boldsymbol{z}^1$
$\boldsymbol{z}(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$
$\boldsymbol{z}^2$
(a)
$\boldsymbol{z}^1$
$\boldsymbol{z}(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$
$\boldsymbol{z}^2$
(b)
$\boldsymbol{z}^1$
$\bar{\boldsymbol{z}} = \underline{\boldsymbol{z}} = \boldsymbol{z}(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$
$\boldsymbol{z}^2$
(c)
Figure 3.11 – $\boldsymbol{z}(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$ is far away from the orthogonal sides of the triangle

We first explain Case A. An illustration of this case is shown in Figure 3.11. For Case A, the operation first calls Find-NDP$\big(\boldsymbol{z}(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)\big)$ to find an efficient solution based on $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$. We denote this efficient solution by $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$ since $f(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$ is an upper bound for the optimal objective value of Problem (3.2). An illustration of $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$ in the criterion space can be found in Figure 3.11b. Obviously, $\boldsymbol{z}(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \in T(\boldsymbol{z}^1, \boldsymbol{z}^2)$. However, we must have that $\boldsymbol{z}(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \neq \boldsymbol{z}^1$ and $\boldsymbol{z}(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \neq \boldsymbol{z}^2$ and this is exactly the consequence of the fact that $z_1(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) < z_1^2 - (z_1^2 - z_1^1)\varepsilon_3$ and $z_2(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) < z_2^1 - (z_2^1 - z_2^2)\varepsilon_3$. The operation then sets $\bar{\boldsymbol{z}} = \underline{\boldsymbol{z}} = \boldsymbol{z}(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$. Note that this implies that Algorithm 3 will later split the triangle from the point $\boldsymbol{z}(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$ (see Figure 3.11c). So, there is no need to split the triangle in the standard way that TSM employs (see subsection 3.2.2).

Finally, we note that the operation also needs to compute lower bounds (for the new rectangles). Obviously, $\boldsymbol{z}(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \neq \boldsymbol{z}(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$ then the operation immediately calls LB-Finder-Rectangle $(\boldsymbol{z}^1, \underline{\boldsymbol{z}})$ and LB-Finder-Rectangle $(\bar{\boldsymbol{z}}, \boldsymbol{z}^2)$ to compute $\underline{L}$ and $\bar{L}$, respectively. Otherwise, there is no need to call them since by definition $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$ is a lower bound solution which is happened to be an efficient solution as well. So, there is no need to split the triangle for further investigation because $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$ is an optimal solution of Problem (3.2) when its search is restricted to the search region defined by $T(\boldsymbol{z}^1, \boldsymbol{z}^1)$. Consequently, in this case, the operation simply sets $\bar{L} = f(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$, $\underline{L} = f(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$ and $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) = (\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$. Note that by doing so, Algorithm 3 will not add new rectangles to the priority queue (see Lines 34-40).

We now explain Case B. In this case, the operation attempts to split the triangle using the standard way that TSM employs (see subsection 3.2.2). This implies that if *direction* = *horizontal* then the operation calls Split-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2, horizontal$) to compute $(\boldsymbol{x}_I', \boldsymbol{x}_C')$ and $(\boldsymbol{x}_I'', \boldsymbol{x}_C'')$, and then it changes the *direction* to 'vertical' (since as we explained in subsection 3.2.2, the direction should alternate in TSM). Similarly, if *direction* = *vertical* then the operation calls Split-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2, vertical$) to compute $(\boldsymbol{x}_I', \boldsymbol{x}_C')$ and $(\boldsymbol{x}_I'', \boldsymbol{x}_C'')$, and then it changes the *direction* to 'horizontal'.

Note that $(\boldsymbol{x}_I', \boldsymbol{x}_C')$ and $(\boldsymbol{x}_I'', \boldsymbol{x}_C'')$ are efficient solutions. So, next the operation calls UB-Finder-Point $\big(\boldsymbol{z}(\boldsymbol{x}_I', \boldsymbol{x}_C')\big)$ to compute an upper solution denoted by $(\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C)$. Also, it calls UB-Finder-Point $\big(\boldsymbol{z}(\boldsymbol{x}_I'', \boldsymbol{x}_C'')\big)$to compute an upper solution denoted by $(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C)$ if

$\boldsymbol{z}(\boldsymbol{x}'_I, \boldsymbol{x}'_C) \neq \boldsymbol{z}(\boldsymbol{x}''_I, \boldsymbol{x}''_C)$. Otherwise, it simply sets $(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C) = (\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C)$. After computing $(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C)$ and $(\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C)$, the operation sets $\bar{\boldsymbol{z}} = \boldsymbol{z}(\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C)$ and $\underline{\boldsymbol{z}} = \boldsymbol{z}(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C)$. The operation then computes the best upper bound solution as follows:

$$(\boldsymbol{x}^u_I, \boldsymbol{x}^u_C) \in \arg\min\{f(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C), f(\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C)\}.$$

Finally, we again note that the operation also needs to compute lower bounds (for the new rectangles). If $\underline{\boldsymbol{z}} \neq \boldsymbol{z}^1$ then the operation immediately calls LB-Finder-Rectangle $(\boldsymbol{z}^1, \underline{\boldsymbol{z}})$ to compute $\underline{L}$. Otherwise, it sets $\underline{L} = f(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C)$ since, by construction, $(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C)$ should provide a lower bound. Similarly, if $\bar{\boldsymbol{z}} \neq \boldsymbol{z}^2$ then the operation calls LB-Finder-Rectangle $(\bar{\boldsymbol{z}}, \boldsymbol{z}^2)$ to compute $\bar{L}$. Otherwise, it sets $\bar{L} = f(\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C)$.

A detailed description of Explore-Triangle $\big((\boldsymbol{x}^l_I, \boldsymbol{x}^l_C), \boldsymbol{z}^1, \boldsymbol{z}^2, \mathit{direction}\big)$ is shown in Algorithm 2. We now make one final comment to avoid redundant calculations:

- For Case B, before calling LB-Finder-Rectangle $(\boldsymbol{z}^1, \underline{\boldsymbol{z}})$, we first need to check whether $\boldsymbol{z}(\boldsymbol{x}^l_I, \boldsymbol{x}^l_C) \in R(\boldsymbol{z}^1, \underline{\boldsymbol{z}})$. If that is the case then we know that $\underline{L} = f(\boldsymbol{x}^l_I, \boldsymbol{x}^l_C)$. Similarly, before calling LB-Finder-Rectangle $(\bar{\boldsymbol{z}}, \boldsymbol{z}^2)$, we first need to check whether $\boldsymbol{z}(\boldsymbol{x}^l_I, \boldsymbol{x}^l_C) \in R(\bar{\boldsymbol{z}}, \boldsymbol{z}^2)$. If that is the case then we know that $\bar{L} = f(\boldsymbol{x}^l_I, \boldsymbol{x}^l_C)$.

3.3.3 Performance of the Algorithm: A Small Example

In this section, we use the example introduced by Belotti et al. [38] to show the performance of the algorithm in terms of the proportion of the nondominated frontier explored

Algorithm 2: Explore-Triangle $\big((\boldsymbol{x}_I^l, \boldsymbol{x}_C^l), \boldsymbol{z}^1, \boldsymbol{z}^2, direction\big)$

```
1  if $z_1(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) < z_1^2 - (z_1^2 - z_1^1)\varepsilon_3$ and $z_2(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) < z_2^1 - (z_2^1 - z_2^2)\varepsilon_3$ then
2      $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \leftarrow$ Find-NDP $\big(\boldsymbol{z}(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)\big)$
3      $\bar{\boldsymbol{z}} \leftarrow \boldsymbol{z}(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$; $\underline{\boldsymbol{z}} \leftarrow \boldsymbol{z}(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$
4      if $\boldsymbol{z}(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) = \boldsymbol{z}(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$ then
5          $\bar{L} \leftarrow f(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$; $\underline{L} \leftarrow f(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$; $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \leftarrow (\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$
6      else
7          $\underline{L} \leftarrow$ LB-Finder-Rectangle $(\boldsymbol{z}^1, \underline{\boldsymbol{z}})$
8          $\bar{L} \leftarrow$ LB-Finder-Rectangle $(\bar{\boldsymbol{z}}, \boldsymbol{z}^2)$
9  else
10     if $direction = horizontal$ then
11         $\big((\boldsymbol{x}'_I, \boldsymbol{x}'_C), (\boldsymbol{x}''_I, \boldsymbol{x}''_C)\big) \leftarrow$ Split-Triangle$(\boldsymbol{z}^1, \boldsymbol{z}^2, horizontal)$
12         $direction \leftarrow vertical$
13     else
14         $\big((\boldsymbol{x}'_I, \boldsymbol{x}'_C), (\boldsymbol{x}''_I, \boldsymbol{x}''_C)\big) \leftarrow$ Split-Triangle$(\boldsymbol{z}^1, \boldsymbol{z}^2, vertical)$
15         $direction \leftarrow horizontal$
16     $(\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C) \leftarrow$ UB-Finder-Point $\big(\boldsymbol{z}(\boldsymbol{x}'_I, \boldsymbol{x}'_C)\big)$
17     if $\boldsymbol{z}(\boldsymbol{x}'_I, \boldsymbol{x}'_C) \neq \boldsymbol{z}(\boldsymbol{x}''_I, \boldsymbol{x}''_C)$ then
18         $(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C) \leftarrow$ UB-Finder-Point $\big(\boldsymbol{z}(\boldsymbol{x}''_I, \boldsymbol{x}''_C)\big)$
19     else
20         $(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C) \leftarrow (\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C)$
21     $\bar{\boldsymbol{z}} \leftarrow \boldsymbol{z}(\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C)$
22     $\underline{\boldsymbol{z}} \leftarrow \boldsymbol{z}(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C)$
23     $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \leftarrow \arg\min\{f(\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C), f(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C)\}$
24     if $\underline{\boldsymbol{z}} = \boldsymbol{z}^1$ then
25         $\underline{L} \leftarrow f(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C)$
26     else
27         $\underline{L} \leftarrow$ LB-Finder-Rectangle $\big(\boldsymbol{z}^1, \underline{\boldsymbol{z}}\big)$
28     if $\bar{\boldsymbol{z}} = \boldsymbol{z}^2$ then
29         $\bar{L} \leftarrow f(\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C)$
30     else
31         $\bar{L} \leftarrow$ LB-Finder-Rectangle $\big(\bar{\boldsymbol{z}}, \boldsymbol{z}^2\big)$
32 return $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u), (\underline{\boldsymbol{z}}, \underline{L}), (\bar{\boldsymbol{z}}, \bar{L}), direction$
```

at the moment an optimal solution was returned:

$$
\begin{aligned}
\max\ & z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) = -3x_1 - 6x_2 + 3x_3 + 5x_4 \\
\max\ & z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) = 15x_1 + 4x_2 + x_3 + 2x_4 \\
\text{s.t.}\ & -x_1 + 3x_5 \le 0 \\
& x_1 - 6x_5 \le 0 \\
& -x_2 + 3x_5 \le 0 \\
& x_2 - 6x_5 \le 0 \\
& -x_3 + 4x_6 \le 0 \\
& x_3 - 4.5x_6 \le 0 \\
& -x_4 + 4x_6 \le 0 \\
& x_4 - 4.5x_6 \le 0 \\
& x_5 + x_6 \le 5 \\
& x_1,\ x_2,\ x_3,\ x_4 \in \mathbb{R}_+ \\
& x_5,\ x_6 \in \mathbb{Z}_+.
\end{aligned}
$$

Note that because the proposed algorithm works with $z_1(\boldsymbol{x}_I, \boldsymbol{x}_C)$ and $z_2(\boldsymbol{x}_I, \boldsymbol{x}_C)$ in minimization form, the coefficients of the objective functions are multiplied by -1 during the course of the algorithm. However, for consistency with Belotti et al. [38], we report their values as if we are solving the maximization form in all figures in this section.

The (exact) nondominated frontier of the presented BOMILP is shown in Figure 3.12. Now suppose that we want to solve $\min_{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}_E} f(\boldsymbol{x}_I, \boldsymbol{x}_C)$ where $\mathcal{X}_E$ is the set of efficient solutions of the presented BOMILP. We constructed four different settings in which the only difference between them is the linear function, i.e., $f(\boldsymbol{x}_I, \boldsymbol{x}_C)$.

For the first three settings, the coefficients of $f(\boldsymbol{x}_I, \boldsymbol{x}_C)$ are drawn randomly from the discrete uniform distribution from the interval $[-100, 100]$ for both continuous and integer

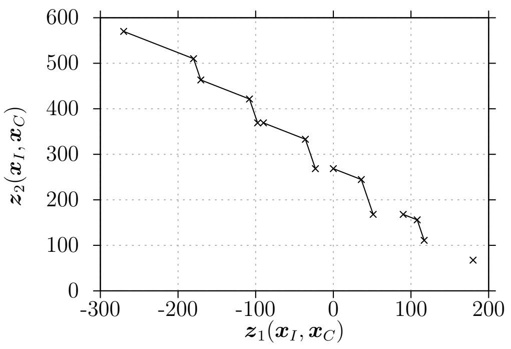


Figure 3.12 – The exact nondominated frontier of the example

variables. For the fourth setting, we set $f(\boldsymbol{x}_I, \boldsymbol{x}_C) = z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) + z_2(\boldsymbol{x}_I, \boldsymbol{x}_C)$. Note that we considered this linear function since minimizing $f(\boldsymbol{x}_I, \boldsymbol{x}_C)$ becomes contradictory to the nature of problem (in which $z_1(\boldsymbol{x}_I, \boldsymbol{x}_C)$ and $z_2(\boldsymbol{x}_I, \boldsymbol{x}_C)$ should be maximized). So, we expect that such a problem to be harder to solve. The partial nondominated frontiers generated during the course of the proposed algorithm for Settings 1 to 4 are shown in Figures 3.13a to 3.13d, respectively. In these figures, we also report $f(\boldsymbol{x}_I, \boldsymbol{x}_C)$, its optimal value (denoted by f^*), and the image of the obtained optimal solution in the criterion space. The latter is shown by a triangle symbol.

Observe that in all settings, only a proportion of the true nondominated frontier is explored during the course of algorithm. Of course, this is precisely what we hoped to achieve (in practice) by the proposed algorithm. Also, not surprisingly, for Setting 4, we see that the a larger proportion of the nondominated frontier is explored since the algorithm obtains weak lower/dual bounds when $f(\boldsymbol{x}_I, \boldsymbol{x}_C) = z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) + z_2(\boldsymbol{x}_I, \boldsymbol{x}_C)$.

3.3.4 Implementation Issues and Enhancements

Since our proposed algorithm is based on TSM, the implementation tips described in Boland et al. [1] should be employed. We now introduce the most important ones, but interested readers may refer to Boland et al. [1] for further details.

- *Tip 1:* Adding as few objective function based constraints as possible to the (single-objective) MILP formulations (arising during the course of the algorithm) often

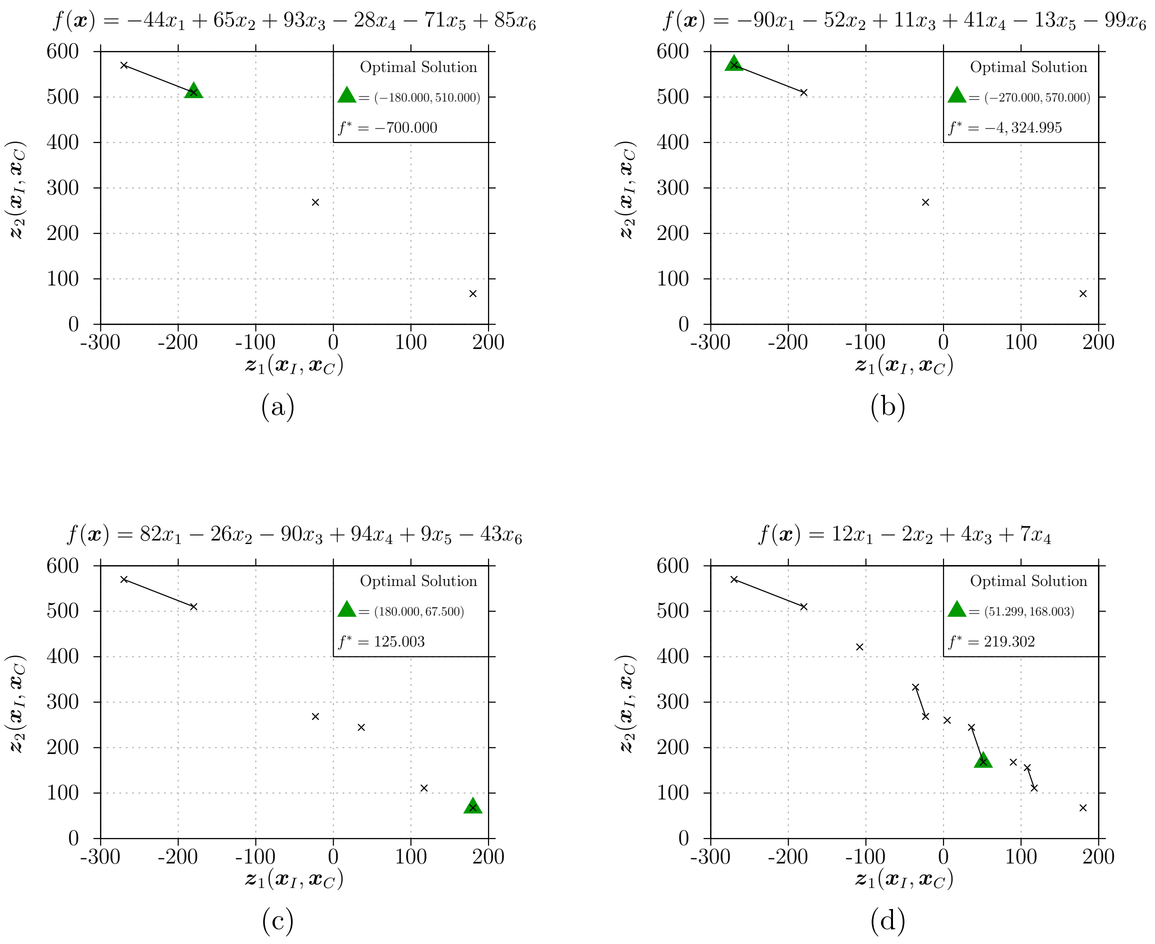


Figure 3.13 – The partial nondominated frontier of the example generated under each setting

reduces the numerical issues and improves the performance of the proposed algorithm in practice. So, in the proposed algorithm, we can replace any instance of $\boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in R(\boldsymbol{z}^1, \boldsymbol{z}^2)$ or $\boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ by:

$$z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq z_1^2$$
$$z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq z_2^1.$$

This is possible since we always have that $\boldsymbol{z}^1, \boldsymbol{z}^2 \in \mathcal{Y}_N$. Similarly, $\boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in H(\boldsymbol{z}^1, \boldsymbol{z}^2)$ can be simplified to:

$$
\begin{gathered}
z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq z_1^2 \\
z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq z_2^1 \\
\lambda_1 z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) + \lambda_2 z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq \lambda_1 z_1^1 + \lambda_2 z_2^1
\end{gathered}
$$

- *Tip 2:* Providing (high-quality) initial feasible solutions to a MILP solver (whenever possible) often improves the performance of the proposed algorithm in practice. One of the feature of the TSM is that it naturally can provide a feasible solution for solving any (single-objective) MILP (after finding $\boldsymbol{z}^T$ and $\boldsymbol{z}^B$) that arises during its course. The same feature is also true for the proposed algorithm. While exploring $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ or $R(\boldsymbol{z}^1, \boldsymbol{z}^2)$, the solution corresponding to $\boldsymbol{z}^1$ or $\boldsymbol{z}^2$ (or both) are feasible and so they can be provided. In general, all operations defined in this study take some points in the criterion space as inputs. So, we can provide the solutions corresponding to those points to (single-objective) MILP solvers.

Next, we present some enhancements and we later in subsection 3.3.5 show their effectiveness in practice.

Enhancement 1 (E$_1$): When calling the operations Split-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2, horizontal$) and Split-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2, vertical$), we can sometimes avoid solving some of the optimization problems. Specifically, we know that the first step of Split-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2, horizontal$) is to solve the following lexicographic operation:

$$
(\boldsymbol{x}'_I, \boldsymbol{x}'_C) \in \arg \operatorname*{lex\,min}_{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}} \{z_1(\boldsymbol{x}_I, \boldsymbol{x}_C), z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) : z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq \frac{z_2^1 + z_2^2}{2}, \boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in T(\boldsymbol{z}^1, \boldsymbol{z}^2)\}.
$$

Note that during the course of the algorithm, we are always in the situation that $\boldsymbol{z}^1$ are $\boldsymbol{z}^2$ are nondominated points. Consequently, if after solving the first optimization problem in the above lexicographic operation, the optimal objective value is equal to z_1^2

then we do not need to solve the second optimization problem since we must have that $\boldsymbol{z}(\boldsymbol{x}'_I, \boldsymbol{x}'_C) = \boldsymbol{z}^2$. Also, we know that the second step of Split-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2, horizontal$) is to solve the following lexicographic operation:

$$(\boldsymbol{x}''_I, \boldsymbol{x}''_C) \in \underset{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}}{\arg \operatorname{lex} \min} \{z_2(\boldsymbol{x}_I, \boldsymbol{x}_C), z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) : z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) < z_1(\boldsymbol{x}'_I, \boldsymbol{x}'_C), \\ \boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in T(\boldsymbol{z}^1, \boldsymbol{z}^2)\}.$$

Consequently, if after solving the first optimization problem in the above lexicographic operation, the value of the optimal objective value is equal to z_2^1 then we do not need to solve the second optimization problem since we must have that $\boldsymbol{z}(\boldsymbol{x}''_I, \boldsymbol{x}''_C) = \boldsymbol{z}^1$. The above observations are also valid for Split-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2, vertical$).

Enhancement 2 (E$_2$): Observe that the intersection of any two elements (which are either triangles or rectangles) in the priority is either empty or a corner point, i.e., $\boldsymbol{z}^1$ or $\boldsymbol{z}^2$. Consequently, for a given corner point, e.g., $\boldsymbol{z}^1$, calling the operation UB-Finder-Point($\boldsymbol{z}^1$) may be redundant (if such an operation should be called at all). Consequently, before calling UB-Finder-Point($\boldsymbol{z}^1$), we propose to check whether such an operation has been called before. If the answer is "yes" then we can simply avoid calling UB-Finder-Point($\boldsymbol{z}^1$).

Enhancement 3 (E$_3$): Right before calling Line-Detector($\boldsymbol{z}^1, \boldsymbol{z}^2$), we propose to always check whether all integer variables in the corresponding solutions to $\boldsymbol{z}^1$ and $\boldsymbol{z}^2$ (that the algorithm maintains in the memory) have the same value. If that is the case then $H(\boldsymbol{z}^1, \boldsymbol{z}^2)$ is entirely part of the nondominated frontier. So, we can avoid solving Line-Detector($\boldsymbol{z}^1, \boldsymbol{z}^2$).

Enhancement 4 (E$_4$): Our numerical experiments show that Line-Detector($\boldsymbol{z}^1, \boldsymbol{z}^2$) is an expensive operation, and so it is better to postpone it. Let n_I be the number of integer variables with same values in the corresponding solutions to $\boldsymbol{z}^1$ and $\boldsymbol{z}^2$ (that the algorithm maintains in the memory). We propose to call Line-Detector($\boldsymbol{z}^1, \boldsymbol{z}^2$) only if $n_I \geq 0.9n_1$. Note that if we do not call this operation, we simply assume that $H(\boldsymbol{z}^1, \boldsymbol{z}^2)$ does not partially or entirely belong to the nondominated frontier.

3.3.5 A Computational Study for the First Implementation of the Algorithm

We conduct a comprehensive computational study to evaluate the performance of the new algorithm. We use the C++ programming language to implement the proposed approach, and employ CPLEX 12.7 as the single-objective mixed integer linear programming solver. All computational experiments are carried out on a Dell PowerEdge R630 with two Intel Xeon E5-2650 2.2 GHz 12-Core Processors (30MB), 128GB RAM, and the RedHat Enterprise Linux 6.8 operating system. We only use a single thread for all experiments.

To test the performance of the proposed algorithm, we modify 25 BOMILPs used in [1] to generate instances for Problem (3.2). It is worth mentioning that these 25 BOMILPs are divided into 5 classes (each contains 5 BOMILPs). Each class is denoted by Cm where m is the number of constraints of each BOMILP in that class. The number of variables and the number of constraints of each instance are equal, i.e., $(n_1 + n_2 = m)$, and half of the variables are binary while the remaining variables are continuous.

From each BOMILP, we generate 10 different instances for Problem (3.2). The only difference between these 10 instances is the linear function, i.e., $f(\boldsymbol{x}_I, \boldsymbol{x}_C)$. The coefficients of the linear function are randomly drawn from the discrete uniform distribution in the interval $[-100, 100]$ for both continuous and integer variables. Note that for each BOMILP, the generated linear functions can be different. Consequently, in total, $25 \times 10 = 250$ instances are tested during this comprehensive study.

The code and instances used in this study can all be found at `https://goo.gl/dixCCi` and `https://goo.gl/DzcCgQ`, respectively. It is worth mentioning that our code is generic and user-friendly in a sense that users only have to provide an input file with a specific format to use our code.

3.3.5.1 Overall Performance

In this section, we show the overall performance of the proposed algorithm. We note that in all results' tables in this section, all enhancements discussed in subsection 3.3.5

are active. Moreover, for each BOMILP/row, averages over the generated 10 instances are reported.

A direct way for optimizing a linear function over the set of efficient solutions of a BOMILP is to first generate the entire nondominated frontier and then applying a post-processing technique to compute an optimal solution. Note that such a post-processing technique is (of course) computationally expensive since multiple optimization problems have to be solved. In light of this observation, we first compare the performance of the our algorithm with TSM in this section. It is clear that if we show that our approach is faster than TSM then the proposed algorithm is of practical value. Note that, in this study, we do not consider the time that has to be added to the time of TSM because of the post-processing.

In Table 3.1, we present a comparison between the proposed algorithm and TSM where 'Time (sec.)' is the solution time in seconds and '#MILP' is the number of (single-objective) MILPs solved. The last two columns of the table show the percentage of decrease in the solution time and the number of (single-objective) MILPs solved by the proposed algorithm in comparison to TSM. Bold numbers in the last two columns show the instances that our algorithm has a better performance on them. It is also worth mentioning that the C++ implementation of TSM used in this study is exactly the one that is used in Boland et al. [1].

It is observed that the proposed algorithm clearly outperforms TSM in terms of solution time for most instances. Only for small size instances, the solution time of TSM is better. For large size instances, the solution time of the proposed algorithm is around 22% less than the solution time of TSM on average. Overall, in terms of the number of (single-objective) MILPs solved, the proposed algorithm is better. For the largest class of instances, we see around 11% reduction in the number of (single-objective) MILPs solved.

Table 3.2 provides more details about the main operations of the new algorithm, i.e., Algorithm 3. We observe that Line-Detector operation takes the most time to solve a (single-objective) MILP with the overall average of $0.77\frac{\text{sec.}}{\text{\#MILPs}}$. That is the main reason that

Table 3.1 – Performance of the new algorithm in comparison to TSM

Class	**New algorithm**		**TSM**		**% Decrease**	
	Time (sec.)	**#MILP**	**Time (sec.)**	**#MILP**	**Time**	**#MILP**
	0.15	86.3	0.18	70.0	**15.0%**	-23.3%
	0.59	188.8	0.52	171.0	-12.5%	-10.4%
C20	0.56	170.9	0.51	151.0	-10.0%	-13.2%
	0.82	220.3	0.76	200.0	-7.5%	-10.2%
	0.17	87.1	0.18	74.0	**7.2%**	-17.7%
Avg.	0.46	150.7	0.43	133.2	-6.2%	-13.1%
	5.71	710.9	6.99	763.0	**18.3%**	**6.8%**
	1.49	283.9	1.98	318.0	**24.8%**	**10.7%**
C40	2.24	327.6	2.80	351.0	**19.9%**	**6.7%**
	2.50	397.5	2.55	392.0	**1.9%**	-1.4%
	2.58	386.5	2.35	319.0	-9.8%	-21.2%
Avg.	2.91	421.3	3.33	428.6	**12.8%**	**1.7%**
	41.76	1,760.5	48.41	1,700.0	**13.7%**	-3.6%
	20.98	1,028.7	31.75	1,245.0	**33.9%**	**17.4%**
C80	34.47	1,538.7	49.97	1,920.0	**31.0%**	**19.9%**
	45.27	1,881.2	52.01	1,841.0	**13.0%**	-2.2%
	29.40	1,480.7	35.70	1,342.0	**17.7%**	-10.3%
Avg.	34.38	1,538.0	43.57	1,609.6	**21.1%**	**4.5%**
	249.68	2,448.2	294.68	2,470.0	**15.3%**	**0.9%**
	256.88	2,143.8	340.96	2,334.0	**24.7%**	**8.1%**
C160	234.42	2,197.5	312.30	2,260.0	**24.9%**	**2.8%**
	549.73	4,299.2	766.72	4,593.0	**28.3%**	**6.4%**
	267.27	2,521.5	324.97	2,541.0	**17.8%**	**0.8%**
Avg.	311.60	2,722.0	407.93	2,839.6	**23.6%**	**4.1%**
	3,621.63	4,041.5	4,594.85	4,490.0	**21.2%**	**10.0%**
	5,790.73	6,341.2	6,448.02	6,404.0	**10.2%**	**1.0%**
C320	3,992.90	4,448.0	5,519.19	5,167.0	**27.7%**	**13.9%**
	4,757.52	4,967.3	6,652.65	6,016.0	**28.5%**	**17.4%**
	3,449.56	4,117.5	4,611.47	4,865.0	**25.2%**	**15.4%**
Avg.	4,322.47	4,783.1	5,565.24	5,388.4	**22.3%**	**11.2%**

we develop enhancements E_3 and E_4 in subsection 3.3.5. We also note that the second most expensive operation is the Explore-Triangle with the overall average of $0.65\frac{\text{sec.}}{\#\text{MILPs}}$.

3.3.5.2 Performance of Enhancements

In subsection 3.3.4, we described a total of four enhancements to improve the performance of the proposed algorithm. Let 'time ratio' of A (for a particular instance) be the ratio of the solution time of A (for that instance) to the maximum solution time of the new algorithm observed during the entire set of experiments (for that instance). Figure 3.14 shows a comparison between time ratios (for all 250 instances) when a particular enhancement is disabled. The first box, i.e., 'All', refers to a configuration that all enhancements are disabled. In the second box, i.e., 'None', no enhancement is disabled. Boxes labeled E_1, E_2,

Table 3.2 – Basic performance statistics of the operations of the new algorithm

	Total		Weighted-Sum-Method		Line-Detector		UB-Finder-Line		LB-Finder-Triangle		Explore-Triangle	
	Time (sec.)	#MILP	Time (sec.)	#MILP	Time (sec.)	#MILP	Time (sec.)	#MILP	Time (sec.)	#MILP	Time (sec.)	#MILP
	0.15	86.30	0.07	32.70	0.00	0.00	0.02	14.20	0.01	5.50	0.06	29.90
	0.59	188.80	0.23	77.00	0.00	0.00	0.10	32.80	0.04	12.00	0.22	63.00
C20	0.56	170.90	0.12	39.50	0.00	0.10	0.04	10.30	0.07	16.00	0.32	101.00
	0.82	220.30	0.30	88.90	0.00	0.00	0.12	38.30	0.06	14.80	0.34	74.30
	0.17	87.10	0.08	41.40	0.00	0.00	0.02	18.80	0.01	4.00	0.06	18.90
Avg.	**0.46**	**150.68**	**0.16**	**55.90**	**0.00**	**0.02**	**0.06**	**22.88**	**0.04**	**10.46**	**0.20**	**57.42**
	5.71	710.90	2.14	330.30	0.03	2.30	1.25	146.00	0.39	39.40	1.91	188.90
	1.49	283.90	0.57	119.60	0.00	0.00	0.22	50.70	0.14	17.30	0.57	92.30
C40	2.24	327.60	0.81	134.40	0.02	3.60	0.36	55.90	0.19	20.20	0.85	109.50
	2.50	397.50	1.01	198.00	0.00	0.00	0.49	89.90	0.21	18.20	0.79	87.40
	2.58	386.50	0.89	156.10	0.00	0.00	0.39	65.30	0.21	26.20	1.08	134.90
Avg.	**2.91**	**421.28**	**1.08**	**187.68**	**0.01**	**1.18**	**0.54**	**81.56**	**0.23**	**24.26**	**1.04**	**122.60**
	41.76	1,760.50	15.94	872.30	1.32	55.40	10.54	412.40	2.50	75.90	11.45	340.50
	20.98	1,028.70	7.74	497.50	0.59	34.00	4.65	230.90	1.32	44.30	6.68	218.00
C80	34.47	1,538.70	11.16	640.10	1.38	74.40	5.75	269.70	3.12	104.30	13.05	446.20
	45.27	1,881.20	15.73	871.00	1.73	68.90	10.37	403.40	3.03	95.20	14.39	438.70
	29.40	1,480.70	10.62	701.10	1.01	60.10	6.32	331.20	2.06	67.60	9.37	316.70
Avg.	**34.38**	**1,537.96**	**12.24**	**716.40**	**1.20**	**58.56**	**7.52**	**329.52**	**2.40**	**77.46**	**10.99**	**352.02**
	249.68	2,448.20	67.87	949.90	19.13	169.00	43.60	434.10	22.21	170.00	96.78	721.20
	256.88	2,143.80	70.18	842.70	20.61	141.30	44.25	384.20	23.54	149.30	98.25	622.30
C160	234.42	2,197.50	70.89	889.00	16.44	148.10	41.73	401.30	20.13	148.10	85.16	607.00
	549.73	4,299.20	172.91	1,742.10	47.82	266.30	104.47	816.30	41.20	275.70	183.25	1,194.80
	267.27	2,521.50	81.68	1,109.60	19.49	137.30	61.77	509.40	20.17	142.70	84.09	618.50
Avg.	**311.60**	**2,722.04**	**92.71**	**1,106.66**	**24.70**	**172.40**	**59.16**	**509.06**	**25.45**	**177.16**	**109.51**	**752.76**
	3,621.63	4,041.50	879.89	1,325.70	381.81	340.30	529.98	626.50	342.79	340.10	1,486.74	1,404.90
	5,790.73	6,341.20	1,401.19	2,115.90	640.13	520.80	880.67	1,014.80	518.21	519.20	2,350.11	2,166.50
C320	3,992.90	4,448.00	1,042.73	1,514.10	390.98	373.10	574.35	695.20	368.35	372.40	1,616.28	1,489.20
	4,757.52	4,967.30	1,113.97	1,536.10	586.55	457.40	618.36	708.90	461.80	456.70	1,976.49	1,804.20
	3,449.56	4,117.50	778.94	1,334.10	349.39	364.50	497.77	613.10	360.68	363.30	1,462.23	1,438.50
Avg.	**4,322.47**	**4,783.10**	**1,043.34**	**1,565.18**	**469.77**	**411.22**	**620.23**	**731.70**	**410.37**	**410.34**	**1,778.37**	**1,660.66**

E_3, and E_4 refer to configurations that only enhancements E_1, E_2, E_3, and E_4 are disabled, respectively.

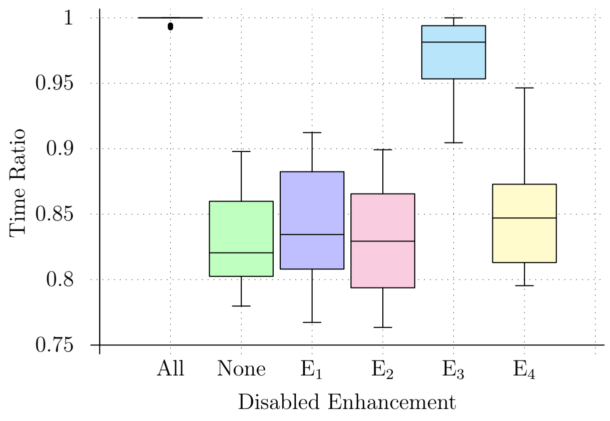


Figure 3.14 – Time ratio boxplot for enhancement techniques

Observe that, on average, the algorithm performs faster when all enhancements are enabled. Also, E_3 is the most effective enhancement for improving the performance of the algorithm. Moreover, disabling all enhancements almost always result in the worst performance of the proposed algorithm. Finally, it seems that enhancements E_1 and E_2 are not contributing much in the performance of the proposed algorithm. To better understand the latter, we next compare '#MILPs ratios'. Let #MILPs ratio of A (for a particular instance) be the ratio of the #MILPs solved by A (for that instance) to the maximum #MILPs solved by the new algorithm (for that instance) during the entire set of experiments.

Figure 3.15 compares #MILPs ratios (for all 250 instances) when a particular enhancement is disabled. Observe that boxes corresponding to None, and E_1 and E_2 are almost identical ('None' is slightly better). That is mainly because that (for our test instances) the algorithm rarely faces a situation that it has to return a corner point of a triangle while splitting it. Overall, from Figures 3.14 and 3.15, we observe that when all enhancements are enabled, both 'Time' and '#MILP' are improved more than 15% compared to the setting that all enhancements are disabled.

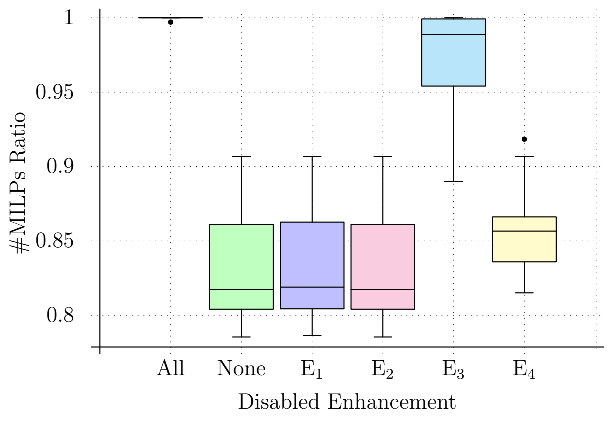


Figure 3.15 – #MILPs ratio boxplot for enhancement techniques

3.3.5.3 Approximate Solutions

We start this section, by showing how the relative optimality gap ratio is changing for an instance of C320 during the course of the algorithm. In Figure 3.16, the horizontal axis shows the iterations of Algorithm 3 and the vertical axis shows the relative optimality gap ratio for an instance of C320. Note that the vertical axis shows ratios, and so, for example, '2' means '200%' (in terms of percentage). We observe that the optimality gap is decreasing reasonably fast during the course of the algorithm. More importantly, the algorithm has found an optimal solution when the gap is approximately 50%. So, basically around half of the iterations of the algorithm are just for proving that the obtained solution is actually optimal.

This particular example indicates that the proposed algorithm probably performs well if the goal is to just generate an approximate solution rather than an optimal solution. In the remaining of this section, we present the results of a series of experiments to indicate that this is actually the case. Note that since our focus is on approximations, we only use the largest class of instances, i.e., C320, in this section.

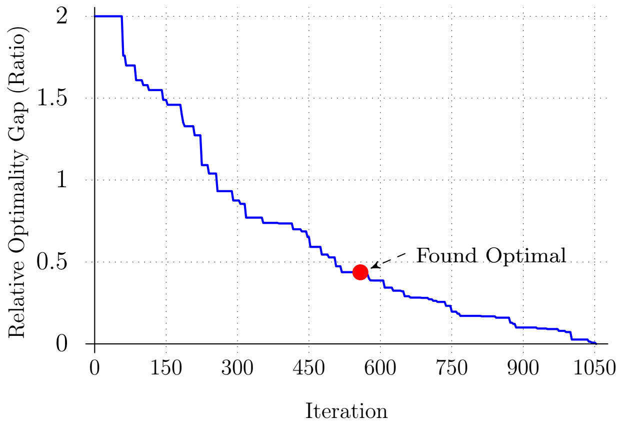


Figure 3.16 – Relative optimality gap obtained in each iteration

3.3.5.3.1 Increasing the Optimality Gap Tolerance

One way to generate an approximate solution is to increase the optimality gap tolerance, i.e., ε_2. The default value for ε_2 is 10^{-5} (which is equivalent to 0.001%). We now test the performance of the algorithm when $\varepsilon_2 \in \{10^{-3}, 10^{-2}, 10^{-1}, 5^{-1}\}$. Note that 5^{-1} is equivalent to 20%. Figure 3.17 shows time ratios (see the definition in subsection 3.3.4) for different values of ε_2.

Observe that there is not much difference between the boxes corresponding to $\varepsilon_2 = 10^{-5}$ and $\varepsilon_2 = 10^{-3}$. In fact, we see that for very few cases, the solution time of $\varepsilon_2 = 10^{-5}$

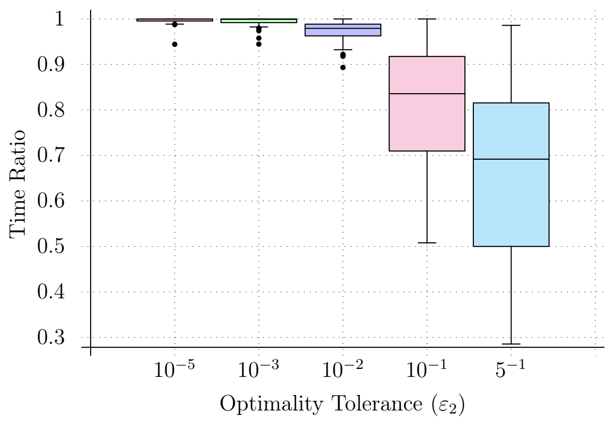


Figure 3.17 – Time ratio for different optimality tolerances

is actually smaller than $\varepsilon_2 = 10^{-3}$, and that is natural since the solution time can fluctuate (when we are running an experiment on a server). Overall, we observe that the solution time improves by around 2% on average by setting $\varepsilon_2 = 10^{-2}$. The improvement percentage in the solution time is around 15% and 30% on average for $\varepsilon_2 = 10^{-1}$ and $\varepsilon_2 = 5^{-1}$.

Let $(\tilde{\boldsymbol{x}}_I, \tilde{\boldsymbol{x}}_C)$ be an approximate solution and $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*)$ be an optimal solution. Note that we treat the solution generated when $\varepsilon_2 = 10^{-5}$ as an optimal solution. We define 'real optimality ratio' as $\frac{f(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*)}{f(\tilde{\boldsymbol{x}}_I, \tilde{\boldsymbol{x}}_C)}$. The real optimality ratio of the generated approximate solutions for different values of ε_2 can be found in Figure 3.18. Observe that the quality of the approximate solution is very close to an optimal solution for all cases. In particular, we observe that even when $\varepsilon_2 = 5^{-1}$, the generated optimal solution is always almost optimal since the real optimality ratio is over 0.995. In fact, there are only very few instances that the approximate solution is up to 3% worse than an optimal solution. Overall, we observe from Figures 3.17 and 3.18 that we can reduce the solution time by 30% and still obtaining an almost optimal solution in practice when $\varepsilon_2 = 5^{-1}$.

3.3.5.3.2 Imposing a Time Limit

Another way of obtaining an approximate solution is to impose a time limit for the proposed algorithm. Let t^* be the time required to compute an optimal solution (i.e., when

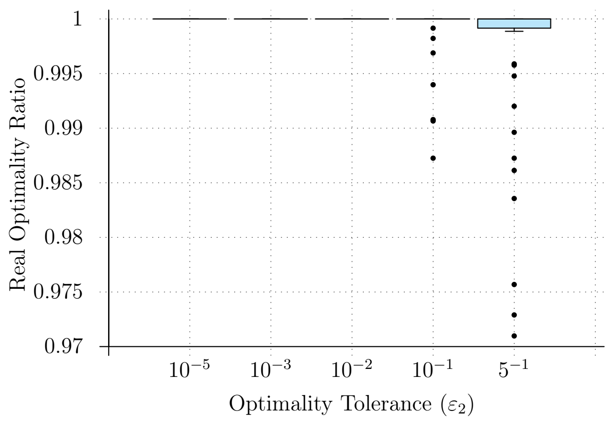


Figure 3.18 – Real optimality ratio for different optimality tolerances

$\varepsilon_2 = 10^{-5}$) for a particular instance. In this section, we study the consequence of imposing a time limit of αt^* where $\alpha \in \{0.05, 0.1, 0.2, 0.4, 0.8\}$ for each instance.

The relative optimality gap ratio that the algorithm is able to reach for different time limits can be found in Figure 3.19. Observe that by just spending around 40% of the total time limit, the relative optimality gap reaches to around 50% on average. Observe too that by spending around 80% of the total time limit, the relative optimality gap reaches to around 10% on average.

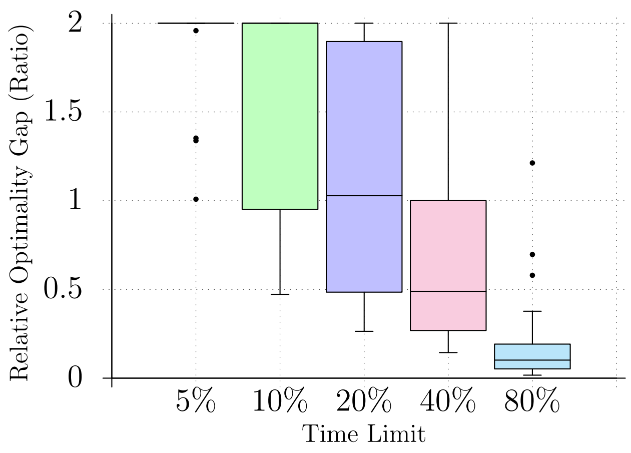


Figure 3.19 – Relative Optimality Gap (Ratio) for different time limits

The real optimality ratio for different time limits can be found in Figure 3.20. Observe that even with the time limit of 5% of the total time, the quality of generated approximate solutions is only around 20% worse than the quality of optimal solutions on average. Observe too that the quality of approximate solutions quickly improves as the time limit increase. In fact, with the time limit of 20% of the total time, the quality of generated approximate solutions is almost equal to the quality of optimal solutions on average. Overall, we observe from Figures 3.19 and 3.20 that by imposing a time limit of 20% of the total time (required to compute an optimal solution), the proposed algorithm reaches to the relative optimality gap of 100%. However, the generated approximate solutions are almost optimal in practice.

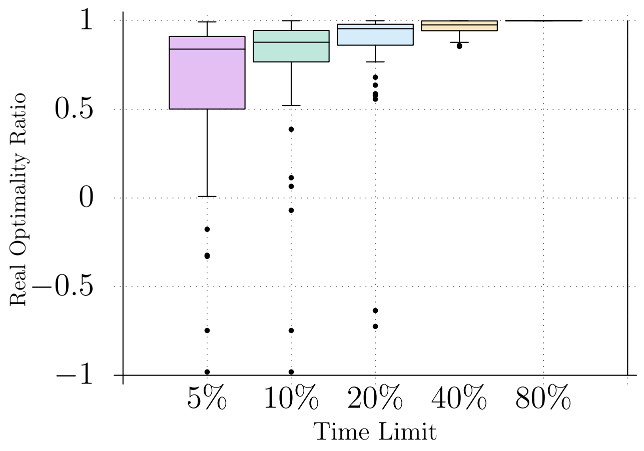


Figure 3.20 – Real optimality ratio for different time limits

3.4 `OOESAlgorithm.jl`: A Julia Package for Optimizing Over the Efficient Set of BOMILP

In this section, we present an extension of our previous work by creating a user-friendly open-source julia package which has the following additional desirable characteristics (compared to its original C++ implementation):

- The package is compatible with the popular `JuMP` modeling language presented in Dunning et al. [39] and supports input in LP and MPS file formats.
- The package supports execution on multiple processors and allows users to choose different parallelization techniques by just tuning a parameter. It is worth mention-

ing that several studies have been conducted about parallelization for evolutionary algorithms in multi-objective optimization (see for instance Pal and Charkhgard [40], and Yu et al. [41]). However, unfortunately, this topic has been almost untouched in the literature of exact algorithms. The recent study conducted by Özlen et al. [42] is one of the few papers (if not the only one) in this scope.

- The package allows to choose between different single-objective optimization solvers by just tuning a parameter. The default solvers include GLPK, CPLEX, Gurobi, Xpress, and SCIP, but it works for all other solvers supported by `MathProgBase.jl` as well.
- The package can be modified by users to return the entire nondominated frontier of a BOMILP.

To the best of our knowledge, there are currently only two documented and supported implementations of multi-objective optimization algorithms in julia (see Gandibleux et al. [43] and Pal and Charkhgard [44]). So, our package contributes to increasing the visibility of multi-objective optimization solvers in julia. In the remaining of this section, our package is referred to as `OOESAlgorithm.jl` and its underlying algorithm as `OOES_Algorithm`.

3.4.1 Main Characteristics of the Package

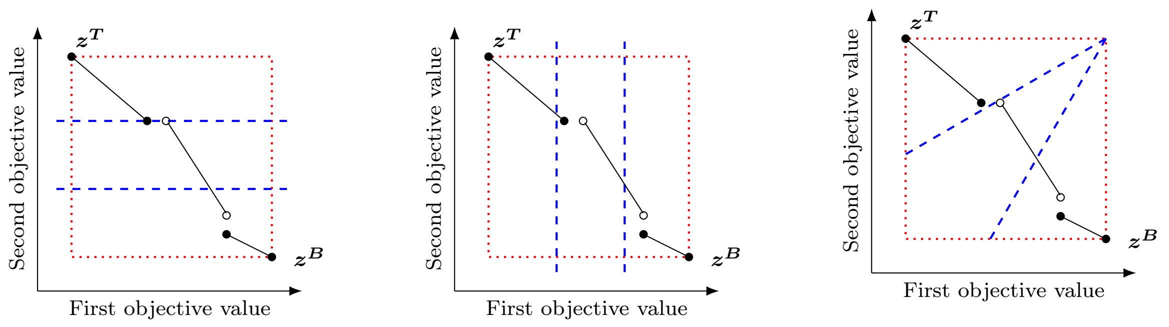
In this section, we detail the main additional characteristics of `OOESAlgorithm.jl` compared to our previous C++ implementation of `OOES_Algorithm` (A comprehensive documentation of the package can be found at Appendix B1).

3.4.1.1 Parallelization Techniques

`OOESAlgorithm.jl` benefits from the recent advances in modern computers, in terms of the number of processors, by exploiting parallelization. The package explores four different parallelization techniques, one of them is based on the priority queue, and the other three are based on decomposing the criterion space.

The simplest and most natural parallelization technique explores elements of the priority queue in parallel using different threads. Suppose that t is the number of available threads for parallelization and q is the number of elements in the priority queue. The first element of the priority queue is assigned to the first available thread to be explored, the second element to the second thread, and this procedure continues until t elements are assigned. We will later show in our numerical experiments that this technique maximizes the utilization of the available threads, i.e., improves the performance of the algorithm significantly.

The criterion space parallelization techniques are based on splitting the unexplored nondominated frontier between the endpoints by adding cuts. We consider three types of cuts to split the criterion space based on their directions including horizontal, vertical, and diagonal. An illustration of these cuts can be found in Figure 3.21 when $t = 3$ (note that the number of cuts is $t - 1$).



(a) Horizontal split (b) Vertical split (c) Diagonal split (origin: (z_1^B, z_2^T))

Figure 3.21 – Splitting directions on the criterion space using three threads

For horizontal splitting, the height of the nondominated frontier is divided by the number of threads that are available for parallelization, i.e., the distance between consecutive cuts is $\frac{z_2^T - z_2^B}{t}$. The following optimization problems are solved to find a nondominated point for each cut $v \in \{1, 2, ..., t-1\}$:

$$(\tilde{\boldsymbol{x}}_I^v, \tilde{\boldsymbol{x}}_C^v) \in \underset{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}}{\arg\min} \; \{z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) : z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq z_2^B + \frac{v(z_2^T - z_2^B)}{t}\},$$

followed by,

$$(\boldsymbol{x}_I^v, \boldsymbol{x}_C^v) \in \underset{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}}{\arg\min} \{z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) + z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) : \boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq \boldsymbol{z}(\tilde{\boldsymbol{x}}_I^v, \tilde{\boldsymbol{x}}_C^v)\}.$$

Observe that, $\tilde{\boldsymbol{z}}^v := \boldsymbol{z}(\tilde{\boldsymbol{x}}_I^v, \tilde{\boldsymbol{x}}_C^v)$ may not be a nondominated point, therefore, the second operation is performed to find a nondominated point $\boldsymbol{z}^v := \boldsymbol{z}(\boldsymbol{x}_I^v, \boldsymbol{x}_C^v)$. Finally, the unexplored nondominated frontier is split into rectangles defined by every pair of consecutive nondominated points. An illustration of this technique using three threads can be found in Figure 3.22.

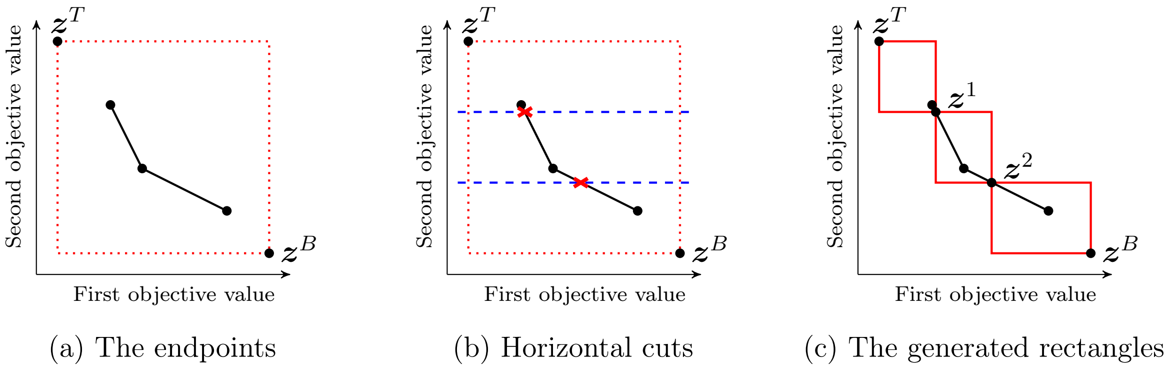


Figure 3.22 – Horizontal splitting of the criterion space using three threads

The vertical splitting technique divides the width of the nondominated frontier by the number of available threads, i.e., the distance between consecutive cuts is $\frac{z_1^B - z_1^T}{t}$. The following optimization problems are solved to find a nondominated point for each cut $v \in \{1, 2, ..., t-1\}$:

$$(\tilde{\boldsymbol{x}}_I^v, \tilde{\boldsymbol{x}}_C^v) \in \underset{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}}{\arg\min} \; \{z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) : z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq z_1^T + \frac{v(z_1^B - z_1^T)}{t}\},$$

followed by,

$$(\boldsymbol{x}_I^v, \boldsymbol{x}_C^v) \in \underset{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}}{\arg\min}\,\{z_1(\boldsymbol{x}_I,\boldsymbol{x}_C) + z_2(\boldsymbol{x}_I,\boldsymbol{x}_C) : \boldsymbol{z}(\boldsymbol{x}_I,\boldsymbol{x}_C) \leq \boldsymbol{z}(\tilde{\boldsymbol{x}}_I^v, \tilde{\boldsymbol{x}}_C^v)\}.$$

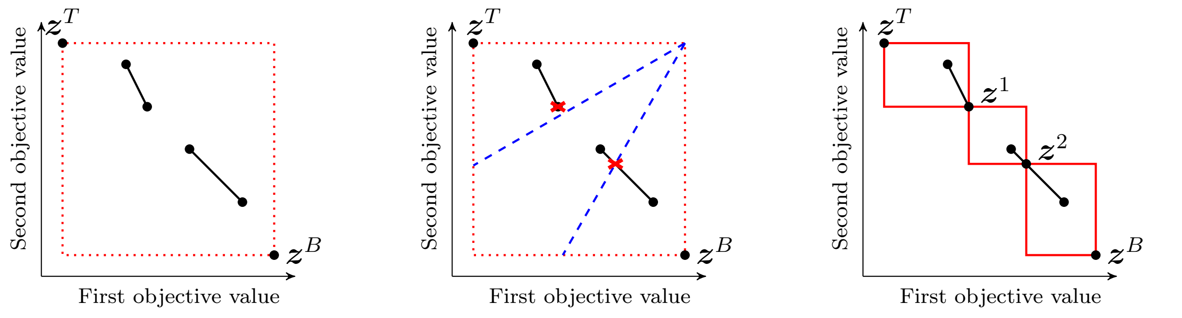
Finally, in the diagonal splitting, the algorithm attempts to divide the criterion space by adding cuts in which their angles are from the set $\alpha \in \{\frac{\pi}{2t}, ..., \frac{(t-1)\pi}{2t}\}$ and originated from the point (z_1^B, z_2^T). The following optimization problems are solved to find a nondominated point for each cut $v \in \{1, 2, ..., t-1\}$:

$$(\tilde{\boldsymbol{x}}_I^v, \tilde{\boldsymbol{x}}_C^v) \in \underset{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}}{\arg\min}\;\{z_2(\boldsymbol{x}_I,\boldsymbol{x}_C) : z_2(\boldsymbol{x}_I,\boldsymbol{x}_C) - z_2^T \geq \tan(\frac{v\pi}{2t})(z_1(\boldsymbol{x}_I,\boldsymbol{x}_C) - z_1^B)\},$$

followed by,

$$(\boldsymbol{x}_I^v, \boldsymbol{x}_C^v) \in \underset{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}}{\arg\min}\,\{z_1(\boldsymbol{x}_I,\boldsymbol{x}_C) + z_2(\boldsymbol{x}_I,\boldsymbol{x}_C) : \boldsymbol{z}(\boldsymbol{x}_I,\boldsymbol{x}_C) \leq \boldsymbol{z}(\tilde{\boldsymbol{x}}_I^v, \tilde{\boldsymbol{x}}_C^v)\}.$$

The first optimization problem finds a feasible solution that minimize the second objective function above the line $z_2(\boldsymbol{x}_I,\boldsymbol{x}_C) - z_2^T = \tan(\frac{v\pi}{2t})(z_1(\boldsymbol{x}_I,\boldsymbol{x}_C) - z_1^B)$. An example of this technique using three threads can be observed in Figure 3.23.



(a) The endpoints (b) Diagonal cuts (c) The generated rectangles

Figure 3.23 – Diagonal splitting of the criterion space using three threads

It is worth mentioning that splitting the criterion space results in at most t independent unexplored rectangles. However, it is possible that different cuts return the same nondominated point, i.e., there may exist $v, w \in \{1, 2, ..., t-1\}$ with $v \neq w$ such that $\boldsymbol{z}^v = \boldsymbol{z}^w$. If that is the case then `OOESAlgorithm.jl` is only able to employ less than t processors. Overall, employing any of the criterion space parallelization techniques will lead to the exploration of at most t independent rectangles. Thus, `OOESAlgorithm.jl` will return at most t independent solutions for Problem (3.2). So, at the time of termination, `OOESAlgorithm.jl` chooses the one that has minimum value for $f(\boldsymbol{x})$.

3.4.1.2 Single-Objective Solvers Supported

Our package benefits from the flexibility of working with julia and the well-known optimization library `MathProgBase.jl` that allows users to choose between different single-objective optimization solvers. The default solvers are GLPK, CPLEX, Gurobi, Xpress, and SCIP, but any solver supported by `MathProgBase.jl` can be employed. Note that, GLPK and SCIP are non-commercial solvers, which means that our package is not limited by license availability.

3.4.2 A Computational Study for the Package

We conduct a comprehensive computational study to evaluate the performance of `OOESAlgorithm.jl`. In the remaining, `OOESAlgorithm.jl` is referred to by the name of the single-objective solver (GLPK 4.61, CPLEX 12.7, Gurobi 8.1, SCIP 6.0.0, and Xpress 8.5) employed in an experiment (by the package). This implies that `GLPK`, `CPLEX`, `GUROBI`, `SCIP`, and `XPRESS` refer to `OOESAlgorithm.jl` when GLPK 4.61, CPLEX 12.7, Gurobi 8.1, SCIP 6.0.0, and Xpress 8.5 are employed as the single-objective solver, respectively. All computational experiments are carried out on a Dell PowerEdge R630 with two Intel Xeon E5-2650 2.2 GHz 12-Core Processors (30MB), 128GB RAM, and the RedHat Enterprise Linux 6.8 operating system. The user manual of our open-source julia package can be found at `http://eng.usf.edu/~amsierra/documents/Documentation_OOESAlg.pdf`. We note that our package is available in METADATA.jl, so users can easily install and use it after reading the user manual. To test the performance of the package, we use the 250 (random) instances employed in subsection 3.3.5.

3.4.2.1 OOESAlgorithm.jl Versus C++ Implementation

The goal of this subsection is to only provide some numerical evidences that a julia implementation can be as good as a C++ implementation in terms of the solution time. So, in this subsection, we show the overall performance of `OOESAlgorithm.jl` compared to our previous C++ implementation of `OOES_Algorithm` (see subsection 3.3.5). Since the C++ implementation works with CPLEX, we use `CPLEX` (which is basically `OOESAlgorithm.jl` when CPLEX is employed) for comparison in this section. In Table 3.3, we present a comparison between `CPLEX` and the C++ implementation where 'Time (sec.)' is the solution time in seconds and '#MILP' is the number of (single-objective) mixed integer linear programs solved. The last two columns of the table show the percentage of decrease in the solution time and the number of (single-objective) MILPs solved by `CPLEX` compared to the C++ implementation. Numbers are averages over 10 instances and bold numbers show the instances for which `CPLEX` has a better performance.

Note that `CPLEX` solves slightly less single-objective mixed integer linear programs. This is due to some very minor enhancements in the implementation of `CPLEX` (that remove some redundant calculations). Overall, we observe that `CPLEX` outperforms C++ implementation for classes C20 and C40 and C80. However, the solution time of these three classes are small to make any meaningful conclusion. We observe that C++ is slightly faster (no more than 3%) than `CPLEX` for larger classes, i.e., C160 and C320, while solving almost the same number of single-objective optimization problems. One possible explanation for this observation could be the fact that the C++ implementation uses the ILOG Concert Technology but `CPLEX` uses `MathProgBase.jl`. To solve an instance of optimization over frontier, multiple single-objective optimization problems should be generated and solved (by our algorithm).

Table 3.3 – Performance of the new algorithm in comparison to C++ implementation

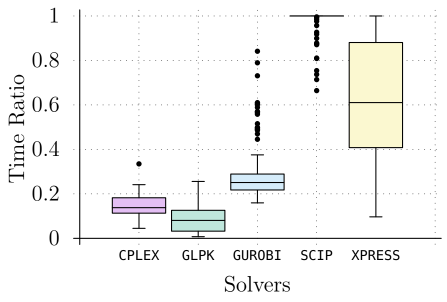
Class	CPLEX		C++		% Decrease	
	Time (sec.)	#MILPs	Time (sec.)	#MILPs	Time (sec.)	#MILPs
C20	0.17	85.40	0.21	85.50	**26.12%**	**0.12%**
	0.55	187.50	0.59	188.30	**5.89%**	**0.43%**
	0.60	167.00	0.66	171.60	**10.16%**	**2.75%**
	0.76	223.00	0.84	224.10	**10.54%**	**0.49%**
	0.17	87.80	0.20	87.80	**21.94%**	**0.00%**
Avg. C20	**0.45**	**150.14**	**0.50**	**151.46**	**11.30%**	**0.88%**
C40	6.06	724.70	6.64	726.00	**9.54%**	**0.18%**
	1.46	280.20	1.61	280.00	**10.42%**	-0.07%
	2.05	311.70	2.33	312.60	**13.69%**	**0.29%**
	2.41	394.60	2.68	395.20	**11.18%**	**0.15%**
	2.64	382.90	2.92	383.50	**10.63%**	**0.16%**
Avg. C40	**2.92**	**418.82**	**3.24**	**419.46**	**10.68%**	**0.15%**
C80	46.52	1,757.60	47.01	1,757.30	**1.06%**	-0.02%
	22.34	1,017.40	23.51	1,017.40	**5.22%**	**0.00%**
	36.82	1,538.60	38.19	1,537.40	**3.72%**	-0.08%
	46.95	1,862.40	49.92	1,863.80	**6.33%**	**0.08%**
	30.88	1,471.80	33.39	1,472.00	**8.11%**	**0.01%**
Avg. C80	**36.70**	**1,529.56**	**38.40**	**1,529.58**	**4.63%**	**0.00%**
C160	281.70	2,457.30	282.50	2,459.00	**0.28%**	**0.07%**
	293.41	2,145.90	290.57	2,145.50	-0.97%	-0.02%
	265.44	2,213.30	266.93	2,213.00	**0.56%**	-0.01%
	632.37	4,290.00	612.31	4,291.20	-3.17%	**0.03%**
	298.77	2,500.30	299.54	2,498.10	**0.26%**	-0.09%
Avg. C160	**354.34**	**2,721.36**	**350.37**	**2,721.36**	-1.12%	**0.00%**
C320	4,083.94	4,022.30	4,009.40	4,016.70	-1.83%	-0.14%
	6,587.88	6,363.90	6,425.72	6,363.90	-2.46%	**0.00%**
	4,440.33	4,329.60	4,308.11	4,332.00	-2.98%	**0.06%**
	5,394.15	4,956.80	5,238.20	4,956.90	-2.89%	**0.00%**
	3,770.92	4,096.30	3,748.88	4,110.50	-0.58%	**0.35%**
Avg. C320	**4,855.44**	**4,753.78**	**4,746.06**	**4,756.00**	-2.25%	**0.05%**

So, ILOG Concert Technology can be possibly faster in doing so because it has been in the market for a longer period of time and has been improved over time.

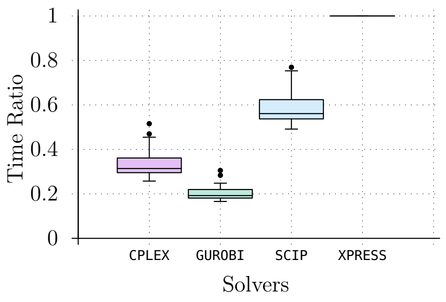
3.4.2.2 Comparison Between Different Solvers

In this subsection, we compare the performance of `OOESAlgorithm.jl` under different single-objective optimization solvers. Figure 3.24 shows *the solution time ratios* of different solvers on different classes of instances. The solution time ratio is the ratio of the solution time to the maximum solution time among all settings in a given figure.

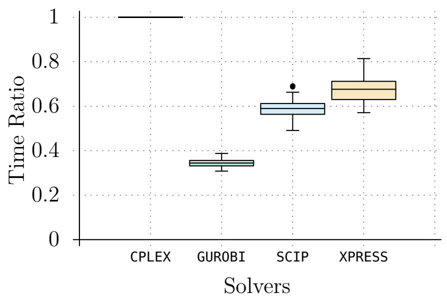
In Figure 3.24a, the time ratios for instances in classes C20, C40 and C80 are reported. Observe that `GLPK` outperforms other solvers even commercial solvers such as `CPLEX`, `GUROBI`, and `XPRESS`. However, as mentioned before, those are small instances in which each single-objective optimization problem takes less than 0.01 to be solved on average. Figures 3.24b and 3.24c show the time ratios for instances in classes C160 and C320, respectively. Note that `GLPK` is not included in these figures since the solver went out of memory when solving such large instances. Observe that `SCIP` performs better than `XPRESS`. However, overall, `GUROBI` seems to be the best solver for solving large instances. From Figure 3.24c, we observe that `GUROBI` is around 60% faster than `CPLEX` for instances of C320. In other words, each single-objective optimization problem is solved around 60% faster on average by Gurobi optimizer. One reason for this observation could be the fact that we have used CPLEX 12.7 which is two years older than Gurobi 8.1.



(a) Instances C20, C40 and C80



(b) Instances C160



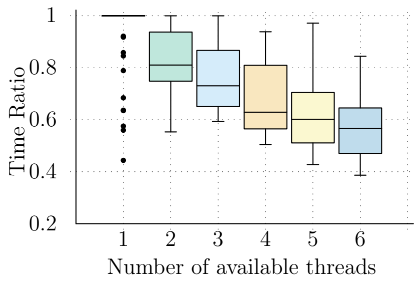
(c) Instances C320

Figure 3.24 – The performance of using different solvers

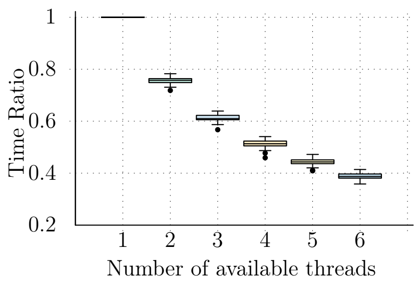
3.4.2.3 Parallelization

In this subsection, we compare the performance of `GUROBI` when multiple threads are available under different parallelization techniques, i.e., *PriorityQueue*, *Horizontal*, *Vertical*, and *Diagonal*. To show the value of parallelization, in this section, only the largest instances, i.e., those in C320, are used. Also, only `GUROBI` is employed in this section because it was shown to perform the best for instances of C320 in the previous section. All our experiments are based on strong scaling, i.e., we compare the results of different number of threads by using fixed size instances.

Figure 3.25 shows a comparison between solution time ratios when $t \in \{1, 2, 3, 4, 5, 6\}$ threads are employed. Specifically, in Figure 3.25a, the box plots of averages of the solution time ratios over all parallelization techniques are reported. Observe that the median of the average time ratios decreases by employing more number of threads in all our experiments. By comparing the medians of the box plot corresponding to 1 and 6 threads, it is evident that the improvement percentage is around 40%. This implies that the median of speedups is almost $\frac{1}{1-0.4} \approx 1.67$ when using 6 threads, i.e., half of the instances are solved at least 1.67 times faster when employing 6 threads.



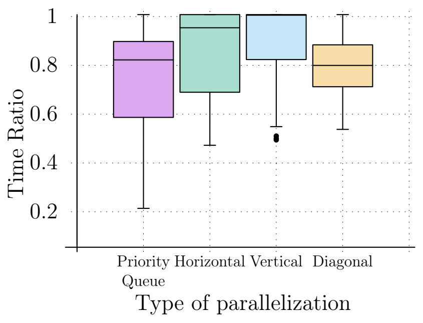
(a) Average of all parallelization techniques



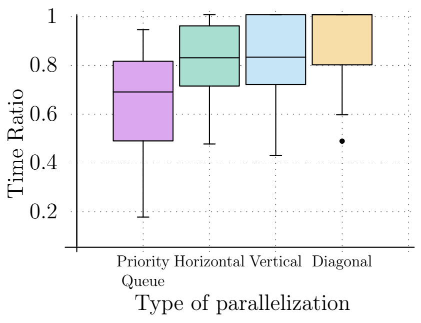
(b) For only *PriorityQueue*

Figure 3.25 – Performance of `GUROBI` when using multiple threads

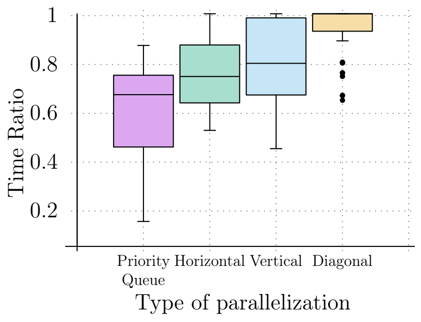
In subsection 3.4.1, we mentioned that *PriorityQueue* maximizes the utilization of the available threads. This can be observed from Figure 3.25b in which it illustrates the box plot of the solution time ratios for different threads when only *PriorityQueue* is used. Observe that the time ratio decreases significantly when more threads are employed. In fact, by comparing the medians of the box plot corresponding to 1 and 6 threads, it is evident that the improvement percentage is around 60%. To highlight the effect of *PriorityQueue* even further, in Figure 3.26, we compare the solution time ratios under different parallelization techniques when $t \in \{2, 3, 4, 5, 6\}$ threads are available. Observe that *PriorityQueue* performs the best and the percentage decrease of the median is between 20% and 50%. This is mainly because in other parallelization techniques, the solution time depends directly on the most difficult rectangle generated after splitting the nondominated frontier. So, the usage of threads can be more unbalanced compared to *PriorityQueue*.



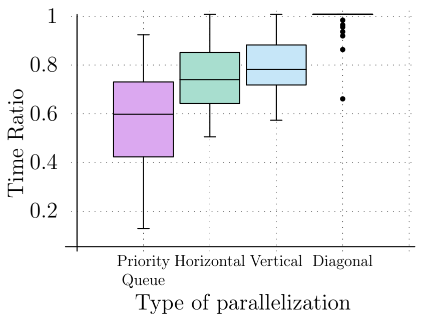
(a) 2 threads available



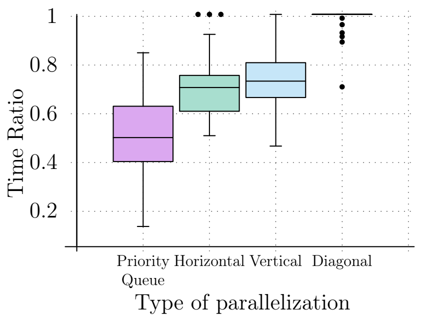
(b) 3 threads available



(c) 4 threads available



(d) 5 threads available



(e) 6 threads available

Figure 3.26 – Performance of `GUROBI` when using multiple threads under different parallelization techniques

To show that our numerical results are not limited to just `GUROBI`, we conducted a set of experiments with `SCIP`. Figure 3.27 shows the time ratios (for all instances) for `SCIP` when multiple threads are available and *PriorityQueue* is employed. Observe that the time ratio decreases significantly when more threads are available. In fact, by comparing the medians of the box plot corresponding to 1 and 6 threads, it is evident that the improvement percentage is around 57%.

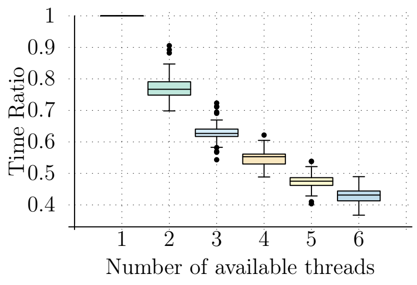


Figure 3.27 – The Performance of `SCIP` when employing *PriorityQueue*

3.5 Conclusion

We presented the first criterion space search algorithm for optimizing a linear function over the set of efficient solutions of BOMILPs. The algorithm is based on TSM and it is easy to implement. We showed that the algorithm is fast and may explore only a small fraction of the nondominated frontier of a BOMILP to compute an optimal solution in practice. We numerically showed that the algorithm converges quickly to an optimal solution and so it is naturally good for computing high-quality approximate solutions.

We also developed `OOESAlgorithm.jl`, a comprehensive open-source user-friendly julia package for optimizing a linear function over the set of efficient solutions for BOMILPs. `OOESAlgorithm.jl` supports execution on multiple processors and exploits different parallelization techniques. It was numerically shown that parallelization helps to improve the solution time significantly. Another desirable characteristic of the package is that it allows users to employ different commercial and non-commercial solvers for solving single-objective optimization problems arising during the course of the algorithm. The computational study showed that even non-commercial solvers can perform quite well. Finally, it was numerically shown that a julia package can be competitive with a C++ package.

Chapter 4: Nash Bargaining Solution: Application of a Special Case of Optimization Over the Efficient Set for Spatial Conservation Planning Problems

In this chapter, we present the third main contribution of this thesis, an application of a special case of optimization over the efficient set, the so-called Nash bargaining solution, for spatial conservation planning problems. We first focus on the ecological motivation behind the spatial conservation planning problem, and the role of operations research in the development of solution approaches. Afterward, we present a bi-objective binary quadratic formulation for the problem that captures the return of investment and the risk in conservation terms. Finally, we introduce a Nash bargaining solution approach for the problem and evaluate its performance.

4.1 Spatial Conservation Planning and the Role of Operations Research

The design of effective protected areas from an ecological and economical point of view is important due to observed declines in biodiversity worldwide (Wand and Önal [45]). Impacts to biodiversity are occurring as a result of habitat loss, fragmentation, species invasions and other threats. These losses are likely to be exacerbated by climate change and the failure of long-term conservation planning to consider the effects of risk and uncertainty (Pressey et al. [46], Reside et al. [47], Eaton et al. [3]). Despite an increase in attention to conservation issues both domestically and internationally, further scientific developments are necessary to support decision making related to the conservation of species and their habitats (Kingsland [48], Boyd et al. [49]). The development of strategies used in the design of nature reserves is often referred to as Spatial Conservation Planning (SCP; Pressey et al.

[46]). SCP can be defined as the process by which conservation programs identify where and how to implement conservation actions (e.g., which unprotected land parcels should be added to a protected reserve based on specific quantifiable criterion to evaluate the value (return) of the parcel as a function of the management objectives) (Schwartz et al. [50], Beyer et al. [51]).

Some of the challenges associated with SCP include: the difficulty of incorporating both our scientific understanding of natural systems and also the values of stakeholders in the decision making process in a way that can be communicated to the non-expert community. To alleviate this problem, SCP has been explored jointly by interdisciplinary teams that involved biologists, economists, and operations researchers, to achieve practical outcomes for the nature reserve design problem (Kingsland [48]). Mathematical models that have been applied to address the SCP problems include network theory (Zamborain-Mason et al. [52]), simulation (Ramage et al. [53]), Markov decision processes (Costello and Polasky [54], Schapaugh and Tyre [55]), and spatial optimization algorithms such as integer (linear) programming and simulated annealing (Önal and Wang [56], Ball et al. [57]). Spatial optimization for SCP uses a mathematical representation of a set of parcels in which the creation of a reserve is desired, and maximizes (or minimizes) one or more objectives, each represented by an equation. When multiple conflicting objectives are available, computing the trade-offs between them (also known as the *Pareto-optimal frontier*) can be helpful for identifying sub-optimal alternatives and focus the trade-off negotiation among the set of optimal reserve designs.

Solving a multi-objective optimization problem is computationally much more time consuming than solving single-objective optimization problems. Consequently, most SCP studies have considered only one objective function with some constraints, often by weighting multiple objectives in a single function. For example, a typical objective is to minimize the total reserve cost subject to one or more conservation goals (see for instance [58, 59, 45, 60, 61, 62, 63, 64]). Some other studies used the return of investment or profit as the objective function and employed some budget constraints (see for instance [65, 66, 67]). Finally, some other authors have used objective functions that consider coverage, connectivity, or distance between parcels as additional evaluation criteria (see for instance [68, 69, 70]).

SCP problems are often modeled as integer programs because the decision of whether a parcel must be added to the reserve requires a binary decision variable. Such optimization problems are often challenging to solve (Beyer et al. [51]) because integer programming is generally NP-hard, i.e., currently there exists no algorithm that can solve any integer programming problem in polynomial time (Garey and Johnson [71]). Despite this limitation, researchers prefer to develop linear integer programs for SCP problems because of the existence of powerful commercial solvers such as IBM ILOG CPLEX, Gurobi, and FICO Xpress, as well as recent advances in the development of exact and heuristic/metaheuristic algorithms. For example, the most widely used software for SCP problems is Marxan, based on a metaheuristic algorithm known as Simulated Annealing (Watts et al. [72]). Marxan uses a standard objective function that aims to minimize the sum of the site-specific and connectivity costs of the selected parcels while reaching objective specific targets (e.g. habitat area targets) in the reserve.

One issue that is often ignored when implementing SCP is the existence of uncertainty in the parameters of the optimization model. A typical approach used to model uncertainty is by replacing unknown parameters with probabilistic estimates. For example the approach described by Tulloch et al. [73], is to determine the number of species protected under different risk tolerance scenarios by defining survival probabilities for the species. Udell et al. [74] used a similar approach for considering collision risk between watercraft and marine wildlife, but they defined scenarios with different survival probabilities for the species using a Bayesian belief network. In a recent study, Haider et al. [75] advised that applying deterministic or stochastic approaches to model SCP problems may either ignore or inaccurately describe uncertainty when the data is not reliable. The authors therefore proposed a robust optimization approach for solving SCP problems to address the issue of uncertainty.

An alternative approach for considering uncertainty in SCP problems is based on Modern Portfolio Theory (MPT), used to assess the trade-off between correlated risk (variance) and expected return. In MPT, the risk function captures the uncertainty inherent in the data. Functionally, MPT is a bi-objective optimization approach in which the first objective is to maximize the return and the second one is to minimize risk. However, there are two challenges when applying MPT:

- Computing the trade-off is computationally expensive for SCP problems involving many parcels. Eaton et al. [3] found it computationally impractical to compute the entire Pareto-optimal frontier for SCPs with more than 50 parcels.

- Even after determining the Pareto-optimal frontier, it is not clear how one should select a desirable solution based on the trade-off.

One of the early applications of MPT for formulating SCP problems was implemented by Halpern et al. [76]. To address the challenges mentioned above, the authors simply combined both objectives into a single objective using a weighted sum function, $\lambda_1 f_1 + \lambda_2 f_2$, where f_1 and f_2 are the two objective functions, and λ_1 and λ_2 are some (positive) scalars (weights). Solving the single-objective weighted sum problem is convenient because it returns a point on the Pareto-optimal frontier of the two objective functions as long as the weights are positive (Aneja and Nair [37]). However, aggregating objective functions is known to have two significant downsides: (1) It is not obvious how λ_1 and λ_2 should be determined, and (2) the number of points on the Pareto-optimal frontier that can be obtained by only using weighted sum and adjusting the weights is often limited, i.e., there are many solutions located on the Pareto-optimal frontier that cannot be obtained by any combination of values for λ_1 and λ_2 (Boland et al. [13]). To illustrate this, observe in Figure 4.1 a Pareto-optimal frontier of two functions f_1 and f_2 with three points P_1, P_2, and P_3. Without loss of generality, we assume that f_1 and f_2 are two minimization functions. Imaginary lines l_1 and l_2 make reference to the (linear) function $\lambda_1 f_1 + \lambda_2 f_2$ for two different combinations of weights λ_1

and λ_2. Note that, for both l_1 and l_2, the values of λ_1 and λ_2 are positive. In the case of l_1, $\lambda_1 \gg \lambda_2$, and for l_2, $\lambda_2 \gg \lambda_1$. The arrows perpendicular to the imaginary lines indicate the direction in which l_1 and l_2 are minimized. Clearly, when l_1 is minimized, P_1 is obtained. Similarly, when l_2 is minimized, P_3 is obtained. However, it is impossible to obtain P_2 from the weighted sum of any combination of positive values for λ_1 and λ_2.

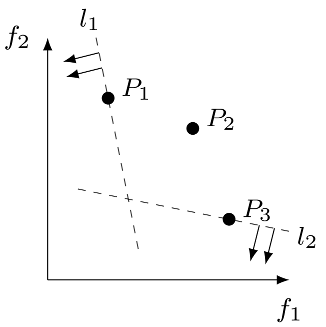


Figure 4.1 – An illustration of a Pareto-optimal frontier with two objective functions and three possible solutions/points.

Recently, Mallory and Ando [77] and Alvarez et al. [78] proposed to only minimize the risk function and include the return function in the set of constraints by imposing some acceptable bounds for it. While promising, this approach has similar weaknesses: (1) It is not obvious how the bounds should be defined, and (2) this approach may return an inferior (i.e., not Pareto-optimal) solution (Boland et al. [13]). More recently, Eaton et al. [3] proposed to first compute the entire trade-off (i.e., the exact Pareto-optimal frontier) and then utilized the concept of cooperative game theory to develop a Nash bargaining solution (Nash [79]) to select a desirable solution from the Pareto-optimal frontier.

A *bargaining problem* is a cooperative game in which all players agree to create a grand coalition, instead of competing, to realize a higher collective payoff (Saghand et al. [80]). To create this grand coalition, an agreement among 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 this question was proposed by Nash, now known as the Nash bargaining solution (Nash [79]). Eaton et al. [3] proposed to create a “game” with two imaginary players in which one of the players wants to minimize the risk and the other one wants to maximize the return. Then, they proposed to create a coalition between these two imaginary players by using the concept of Nash bargaining solution.

The approach proposed by Eaton et al. [3] resolves the second challenge mentioned above regarding MPT. However, the first challenge remains in place. The primary limitation of their approach is that the Pareto-optimal frontier needs to be computed first before applying the Nash bargaining solution. Here, we extend the work proposed by Eaton et al. [3] and show that the Nash bargaining solution can be computed directly without computing the entire Pareto-optimal frontier first. Specifically, we formulate the problem of selecting a Nash bargaining solution in MPT as a Binary Quadratically Constrained Quadratic Program (BQCQP). Through an extensive computational study we show that our approach, able to be implemented with commercial solvers such as CPLEX, Gurobi, and Xpress, can solve problems of up to 800 parcels to approximate optimality in a reasonable time ($\leq$ 8 hours) instead of 50 parcels with the previous approaches. Thus, our extension should considerably increase the applicability of spatial optimization algorithms that consider multi-objective problems. Unlike existing studies in the literature of SCP that mainly focus on linear integer programming approaches, we do not attempt to linearize our proposed BQCQP. We will show that if one wants to linearize the proposed formulation then a significant number of new constraints and decision variables need to be introduced. However, in that case, commercial linear integer programming solvers may struggle to solve problems with such formulation. Because our proposed approach is a mathematical programming formulation, we can estimate how far the best solution obtained from the solver algorithm is relative to a possible optimal solution. The ability to report an optimality gap represents a further another benefit of our method over the one proposed by Eaton et al. [3].

The remainder of the chapter is organized as follows. In Section 4.2, we describe the mathematical formulation of SCP when employing MPT (henceforth referred to as MPT-SCP). In Section 4.3 we detail our proposed BQCQP for solving the MPT-SCP problem using the Nash bargaining solution approach and apply it using a numerical example and a comprehensive computational study. In the last section we provide some conclusions and perspectives.

4.2 Mathematical Formulation of an MPT-SCP Problem

In this section, we present the bi-objective binary quadratic formulation for an MPT-SCP problem and the issues related to such formulation.

4.2.1 Return and Risk Functions

The challenge under consideration is a generic SCP problem that we approach using the principles of MPT (Markowitz [81, 82]), which attempts to maximize the expected return of investment in conservation design while minimizing the risk. The SCP problem is derived from the so-called "reserved selection problem" (Beyer et al. [51]). Specially, the problem assumes that there are two sets of parcels that define the initial state of the system. One is denoted by S and represents the parcels that are already under protection. The other set is denoted by B and represents the parcels that are not already under protection. For any already protected parcel $j \in S$, the decision is whether it should be selected for *sell* (or divestment) to be removed from the protected set of parcels. For any already unprotected parcel $j \in B$, the decision is whether the parcel should be selected for *purchase* (or investment) to be added to the set of protected parcels.

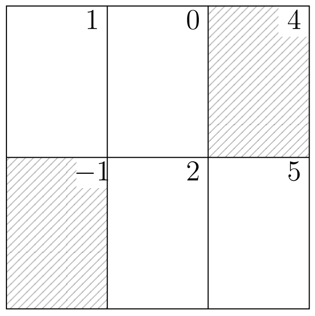
MPT seeks to identify decisions by maximizing the total return for a given level of risk or, alternatively, minimize the risk for a desired level of return. However, the return for each parcel is assumed to be uncertain due to changing conditions, an incomplete understanding of the dynamics of the system being conserved, or for other reasons. Therefore, an uncertain return includes some risk that any decision regarding a reserve design will not achieve its intended outcome. Economic theory suggests that downside risk increases positively with expected value, thus representing an important trade-off for decision makers to consider (Eaton et al. [3]). MPT recognizes this trade-off and seeks to maximize overall expected return while minimizing the total risk (meaning the total variance) at the same time. Moreover, it assumes that there is a correlation between the return obtained from all protected parcels and the risk incurred in their protection. The total expected return of a protected area is computed by adding the expected return of each parcel under protection. We denote the expected return of parcel j by μ_j. To compute the total risk we account for the variance of each parcel being considered for protection (denoted by σ_j^2) but we also consider the correlation between all candidate pairs of parcels (dented by ρ_{ji}). The total expected return and risk can be stated by the following functions:

$$z_1(\boldsymbol{x}, \boldsymbol{y}) = \sum_{j \in B} x_j \mu_j + \sum_{j \in S} (1 - y_j) \mu_j \tag{4.1}$$

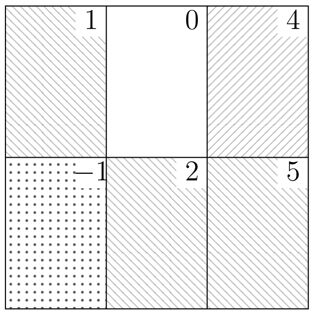
$$\begin{aligned} z_2(\boldsymbol{x}, \boldsymbol{y}) = & \sum_{j \in B} x_j \sigma_j^2 + \sum_{j \in S} (1 - y_j) \sigma_j^2 + \sum_{j \in B} \sum_{i \in B: i > j} 2 x_j x_i \sigma_j \sigma_i \rho_{ji} \\ & + \sum_{j \in S} \sum_{i \in S: i > j} 2(1 - y_j)(1 - y_i) \sigma_j \sigma_i \rho_{ji} + \sum_{j \in B} \sum_{i \in S} 2 x_j (1 - y_i) \sigma_j \sigma_i \rho_{ji} \ , \end{aligned} \tag{4.2}$$

where $x_j \in \{0, 1\}$ for each $j \in B$ is a binary decision variable that indicates if a parcel j from the set B is selected for investment ($x_j = 1$) or not ($x_j = 0$). Similarly, $y_j \in \{0, 1\}$ for all $j \in S$ is a binary decision variable that indicates if a parcel j from the set S is selected for divestment ($y_j = 1$) or not ($y_j = 0$). Values $z_1(\boldsymbol{x}, \boldsymbol{y})$ and $z_2(\boldsymbol{x}, \boldsymbol{y})$ represent the total expected return function and the total risk function, respectively. For $z_1(\boldsymbol{x}, \boldsymbol{y})$, we observe that for any $j \in B$, μ_j will be added to the total expected return only if $x_j = 1$. However, for any $j \in S$, μ_j will be added to the total expected return only if $y_j = 0$. This is because $y_j = 0$ implies that parcel $j \in S$ will remain protected. For $z_2(\boldsymbol{x}, \boldsymbol{y})$, we observe that the risk function has five parts. The first two parts capture the variance of unprotected and protected parcels. The remaining three parts capture the covariance of each pair (i, j) of parcels where both parcels can be unprotected, protected, or one of them is protected and the other is unprotected. Additionally, the covariance implies that $i \neq j$.

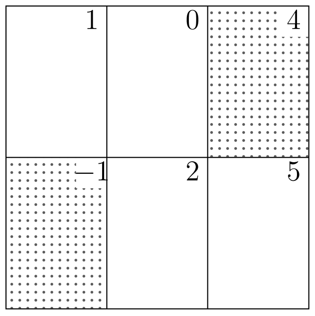
We provide an example of the MPT-SCP problem in Figure 4.2. Figure 4.2a shows the initial system state (current reserve design). A total of 6 parcels exists in this example and only 2 of them are already under protection. The expected return for each parcel is shown in the top-right corner of each parcel and it is assumed that the variance of the return for each parcel is a positive value. Therefore, the current total expected return of the predefined parcels is $4 + (-1) = 3$ and the total risk is some positive number. First suppose that one is interested in optimizing only the return function and ignoring the risk. In this case, some of the unprotected parcels with positive return must be added for protection and the parcels with negative return must be divested from protection, as shown in Figure 4.2b. Hence, the expected maximum total return of the predefined parcels is $1 + 4 + 2 + 5 = 12$. Additionally, from Figure 4.2b, we observe that the parcel with the expected return equal to 0 is not added to the final solution (portfolio). This is because by assumption each parcel has a positive risk. So, adding this parcel does not improve the total expected return but just increases the risk. Now suppose that one is interested in optimizing only the risk function and ignoring the excepted return completely. In this case, again because we assumed that the risk of each parcel is positive, the optimal decision is to divest all already protected parcels because the risk will become zero in that case (and the total return will be zero too). This solution is shown in Figure 4.2c.



(a) Initial state of conservation



(b) Portfolio after optimizing the return



(c) Portfolio after optimizing the risk

☐ Unprotected parcels	▧ Parcels added for protection
▨ Protected parcels	⬚ Parcels divested from protection

Figure 4.2 – An example of MPT-SCP

4.2.2 A Bi-Objective Optimization Formulation

Using the proposed total return and risk functions, the bi-objective optimization formulation corresponding to the MPT-SCP can be stated as follows:

$$\begin{aligned} \max \quad & z_1(\boldsymbol{x}, \boldsymbol{y}) \\ \min \quad & z_2(\boldsymbol{x}, \boldsymbol{y}) \\ \text{subject to:} \quad & (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{F}, \end{aligned} \tag{4.3}$$

where $\mathcal{F} \subseteq \{0,1\}^{|B|+|S|}$ is the so-called *feasible set* of the problem, i.e., the set of all possible/feasible choices for $(\boldsymbol{x}, \boldsymbol{y})$. To define the feasible set, some constraints need to be introduced (unless $\mathcal{F} = \{0,1\}^{|B|+|S|}$). A finite budget is a common example of a constraint, and one that we will use for this study. A budget constraint can be defined in several ways. For example, a monetary budget can be imposed to define a bound on the amount of money available to buy unprotected parcels. Similarly, one can define the maximum number of transactions to bound the number of unprotected parcels that can be bought. In both cases, the funds available and the number of transactions for parcel acquisition can be increased when a currently protected parcel is sold. A generic mathematical representation of a budget constraint can be stated as follows:

$$\sum_{j \in B} \alpha_j x_j - \sum_{j \in S} \beta_j y_j \leq \gamma, \tag{4.4}$$

where γ is a non-negative parameter showing the budget available, α_j for $j \in B$ is a parameter that penalizes the budget when an unprotected parcel j is acquired. Finally, β_j for $j \in S$ is a parameter that increments the budget when a protected parcel j is sold.

In addition to budget constraints, there are several other types of constraints that can appear when dealing with SCP problems. In particular, Beyer et al. [51] formulated constraints to consider the role of adjacent and neighboring parcels, a common concern for the reserve selection problem to address the need for connectivity within a protected area. A compendium of such constraints is provided in Appendix C1 and interested readers are encouraged to refer to Beyer et al. [51] for further details.

Finally, it is worth mentioning that since Problem (4.3) has two competing objectives, there often exists no feasible solution that can optimize both objectives simultaneously. Hence, as mentioned above, a practical approach is to first compute the so-called *Pareto-optimal* frontier, i.e., the set of feasible points in the objectives space that are not dominated by any other feasible point. Next, decision makers select their desired Pareto-optimal point and implement its corresponding reserve portfolio. An illustration of the Pareto-optimal frontier for a MPT-SCP problem is indicated in Figure 4.3. Each point/asterisk in this figure represents a feasible solution in the objectives space. However, asterisks are dominated by at least one of the solid points, i.e., the objective values of at least one of the solid points are better, i.e., the return is higher and risk is lower, than the asterisks. Hence, the solid points describe the Pareto-optimal frontier.

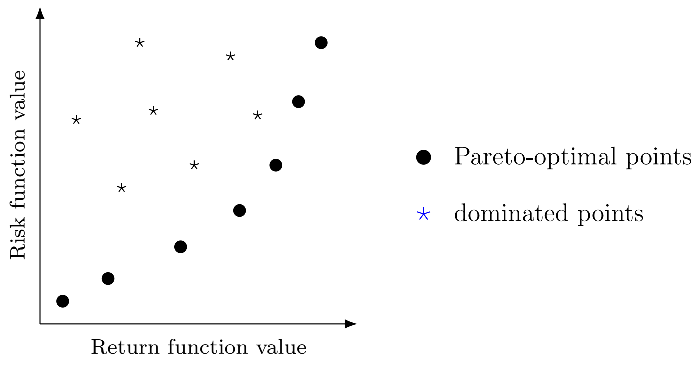


Figure 4.3 – An illustration of the Pareto-optimal frontier of a MPT-SCP problem

4.2.3 Issues Related to the Bi-Objective Optimization Formulation

Although computing the entire Pareto-optimal frontier seems to be a plausible option, there are many barriers and considerations when doing so for practical size SCPs. The primary impediment is related to the nonlinearity of the risk function in Problem (4.3). A common practice for managing nonlinear objectives is linearization (see for instance Eaton et al. [3]). Since all decision variables of Problem (4.3) are binary, standard linearization techniques can be applied easily. For example, we can replace any instance of $x_j x_i$ with a new binary variable denoted by b_{ji} and add the following three constraints,

$$\begin{aligned} &b_{ji} \leq x_j \\ &b_{ji} \leq x_i \\ &b_{ji} \geq x_j + x_i - 1. \end{aligned}$$

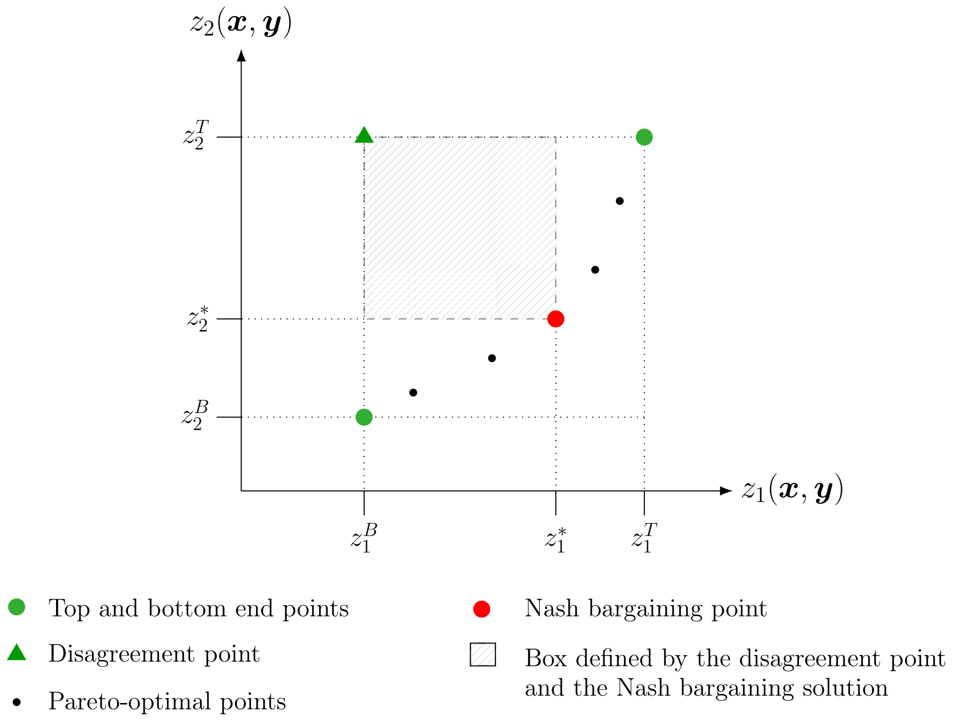
Observe from these inequalities that because b_{ji} is a binary variable, it will take the value of zero if $x_j = 0$ or $x_i = 0$. Otherwise, b_{ji} will take the value of one. A similar approach can be used to linearize any other nonlinear term in the risk function. However, the issue is that for any pair of parcels, there is a nonlinear term in the risk function. So, for each of them, a new binary decision variable and three constraints should be defined. This immediately implies that the size of Problem (4.3) after linearization will be very large for a large number of parcels, e.g., 500 parcels. In this case, it can be expected that even finding a small number of Pareto-optimal points may not be computationally possible.

Here, we propose to estimate and apply the nonlinear risk function directly rather than linearize the objective functions. Overall, commercial linear integer programming solvers are significantly more mature and faster than commercial nonlinear integer programming solvers. However, a desirable feature of the risk function is that it is a quadratic function and commercial nonlinear integer programming solvers have good techniques to deal with such functions. From experience with the current technology and algorithms, it appears to be impractical to compute the Pareto-optimal frontier of Problem (4.3) if $|B| + |S| \geq 50$. As a result, the size of the problems that have been solved to date is quite small. Of course, for larger problems one can divide the total number of parcels into some groups, each with a smaller number of parcels, and compute the Pareto-optimal frontier for each of these groups independently (e.g., Eaton et al. [3]). However, it is clear in that case, the obtained frontiers will be sub-optimal with respect to the full problem.

Another major issue of computing the entire Pareto-optimal frontier is that it is likely to generate a very large number of viable alternative portfolio options, which can be overwhelming for decision makers. In other words, providing many Pareto-optimal points to decision makers may be of limited assistance to guide real decision making (Jorge [16]). One alternative approach is to select the desirable Pareto-optimal point directly for them. Hence, Eaton et al. [3] proposed to automate this procedure and use the Nash bargaining procedure for selecting desirable solution (Nash [79]). They demonstrated that their approach may offer a reasonable starting place for negotiations among decision makers by identifying a portfolio that balances risk and return. However, the main limitation of their approach is that it relies on computing the entire Pareto-optimal frontier in advance, which is not possible computationally for large problems. Hence, in this study, we propose a technique that can find the Nash bargaining solution directly without having first to define the entire Pareto frontier.

4.3 A Nash Bargaining Solution Approach for the MPT-SCP Problem

We begin by explaining the geometric interpretation of the Nash bargaining solution for a MPT-SCP problem using an illustrative example (Figure 4.4). In this Figure, the Pareto-optimal points are shown using solid (small and large) circles. The Nash solution attempts to find a Pareto-optimal point, denoted by $\left(z_1^*, z_2^*\right)$, that the area of the box between this point and a (user-defined) reference point is maximized. The Nash solution is denoted by a red circle and the reference point is shown by a triangle in Figure 4.4. The reference point is sometimes referred to as the disagreement point and shows the status quo (of the game) in Game Theory. A natural choice for the reference point is the so-called *nadir point* (denoted by (z_1^N, z_2^N)) which is basically a point in the criterion space that shows the worst value of each objective function in the Pareto-optimal frontier. This point can be computed for cases with two objectives (such as MPT-SCP problems) by simply finding the endpoints of the Pareto-optimal frontier. Let (z_1^B, z_2^B) and (z_1^T, z_2^T) be the bottom and top endpoints of the Pareto-optimal frontier. The nadir point is $(z_1^N, z_2^N) := (z_1^B, z_2^T)$.



$z_2(\boldsymbol{x}, \boldsymbol{y})$
z_2^T
z_2^*
z_2^B
$z_1(\boldsymbol{x}, \boldsymbol{y})$
z_1^B
z_1^*
z_1^T
● Top and bottom end points
▲ Disagreement point
• Pareto-optimal points
● Nash bargaining point
▨ Box defined by the disagreement point and the Nash bargaining solution

Figure 4.4 – An illustration of the Nash bargaining solution for a MPT-SCP

The Nash bargaining solution is interesting since it intuitively attempts to not only maximizes the benefits of cooperation but also equally distribute it between players. In MPT-SCP, one can consider two imaginary players. The first player attempts to maximize the return and the second player attempts to minimize the risk. Observe that $z_1^* - z_1^N$ is the benefit of cooperation for the first player and $z_2^N - z_2^*$ is the benefit of cooperation for the second player. The Nash bargaining solution attempts to maximize the area of the box, i.e., $(z_1^* - z_1^N)(z_2^N - z_2^*)$. This implies that it geometrically attempts to form a square in the criterion space, i.e., $(z_1^* - z_1^N) \approxeq (z_2^N - z_2^*)$.

Overall, the Nash bargaining solution of Problem (4.3) has the following desirable characteristics. First, it is a Pareto-optimal point, and second, it exists if Problem (4.3) is feasible. In light of this observation, our proposed approach for computing the Nash bargaining solution has three steps. The first two steps are for calculating the reference point (or the nadir point). Specifically, in the first one, the top endpoint should be computed. This can be done by first solving the following optimization problem,

$$z_1^T = \max_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{F}} z_1(\boldsymbol{x},\boldsymbol{y}), \tag{4.5}$$

and then solving the following optimization problem,

$$z_2^T = \min_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{F}} \{z_2(\boldsymbol{x},\boldsymbol{y}) : z_1(\boldsymbol{x},\boldsymbol{y}) = z_1^T\}. \tag{4.6}$$

In the second step, the bottom endpoint should be computed. This can be done by first solving the following optimization problem,

$$z_2^B = \min_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{F}} z_2(\boldsymbol{x},\boldsymbol{y}), \tag{4.7}$$

and then solving the following optimization problem,

$$z_1^B = \max_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{F}} \{z_1(\boldsymbol{x},\boldsymbol{y}) : z_2(\boldsymbol{x},\boldsymbol{y}) = z_2^B\}. \tag{4.8}$$

After computing the endpoints, the nadir point is obtained, i.e., $(z_1^N, z_2^N) := (z_1^B, z_2^T)$. So, in the third step, we can compute the Nash bargaining solution by solving the following optimization problem,

$$\begin{aligned} \max_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{F}} \quad & \big(z_1(\boldsymbol{x},\boldsymbol{y}) - z_1^N\big)\big(z_2^N - z_2(\boldsymbol{x},\boldsymbol{y})\big) \\ \text{subject to} \quad & z_1(\boldsymbol{x},\boldsymbol{y}) \geq z_1^N \\ & z_2(\boldsymbol{x},\boldsymbol{y}) \leq z_2^N. \end{aligned} \tag{4.9}$$

The constraints of Problem (4.9) ensure that the Nash bargaining solution will be on the Pareto-optimal frontier, i.e., it will dominate the reference point. In summary, in our proposed approach, five optimization problem should be solved i.e., (4.5)-(4.9). All these optimization problems can be solved using commercial solvers but there are some implementation issues that should be considered.

4.3.1 Implementation Issues

Problem (4.5) is a binary linear program and commercial solvers can solve it. However, Problem (4.6) is a binary quadratic program since its objective function is a risk function. In order to use commercial solvers to solve Problem (4.6), one can use the following equivalent formulation,

$$z_2^T = \min_{r \geq 0,\ (\boldsymbol{x},\boldsymbol{y}) \in \mathcal{F}} \{r : r \geq z_2(\boldsymbol{x},\boldsymbol{y}), z_1(\boldsymbol{x},\boldsymbol{y}) = z_1^T\}, \tag{4.10}$$

where r is a dummy non-negative variable for capturing the value of the risk in an optimal solution. It is also worth mentioning that an optimal solution of (4.5) is feasible for Problem (4.6). So, we can provide it to the commercial solvers as a warm start solution, i.e., initial solution, to possibly reduce their computational time.

For the bottom endpoints, we propose to skip solving both Problems (4.7) and (4.8) completely. This is because the bottom endpoint attempts to compute the point with the minimum total risk. The minimum possible value for the risk is zero that can be obtained by divesting all parcels that are already under protection. In this case, the total return will be also zero. Hence, we can set $(z_1^B, z_2^B) = (0, 0)$.

Finally, we observe that the objective function of Problem (4.9) is a multiplication between a linear function and a quadratic function. To solve this problem using commercial solvers, we apply a transformation on Problem (4.9) to transform it into a Binary Quadratically Constrained Quadratic Program (BQCQP). In order to do so, we first observe that

Problem (4.9) is equivalent to the following problem,

$$\begin{aligned}\max_{\omega\geq 0,\ (\boldsymbol{x},\boldsymbol{y})\in\mathcal{F}} \quad & \omega\\ \text{subject to} \quad & \omega \leq \sqrt{\big(z_1(\boldsymbol{x},\boldsymbol{y})-z_1^N\big)\big(z_2^N - z_2(\boldsymbol{x},\boldsymbol{y})\big)}\\ & z_1(\boldsymbol{x},\boldsymbol{y}) \geq z_1^N\\ & z_2(\boldsymbol{x},\boldsymbol{y}) \leq z_2^N.\end{aligned}$$

where ω is a dummy non-negative variable capturing the value of the square root of the objective function of Problem (4.9) in an optimal solution. This formulation itself is equivalent to the following formulation,

$$\begin{aligned}\max_{\omega\geq 0,\ r\geq 0,\ (\boldsymbol{x},\boldsymbol{y})\in\mathcal{F}} \quad & \omega\\ \text{subject to} \quad & \omega \leq \sqrt{\big(z_1(\boldsymbol{x},\boldsymbol{y})-z_1^N\big)\big(z_2^N - r\big)}\\ & r \geq z_2(\boldsymbol{x},\boldsymbol{y})\\ & z_1(\boldsymbol{x},\boldsymbol{y}) \geq z_1^N\\ & z_2(\boldsymbol{x},\boldsymbol{y}) \leq z_2^N.\end{aligned}$$

The latter transformation is similar to the one we used in Problem (4.10). In other words, r is a dummy non-negative variable for capturing the value of the risk in an optimal solution. Finally, we observe that this formulation is also equivalent to the following formulation,

$$
\begin{aligned}
\max_{\omega\geq 0,\ \Omega_1\geq 0,\ \Omega_2\geq 0,\ r\geq 0,\ (\boldsymbol{x},\boldsymbol{y})\in\mathcal{F}} \quad & \omega \\
\text{subject to} \quad & \omega^2 \leq \Omega_1\Omega_2 \\
& r \geq z_2(\boldsymbol{x},\boldsymbol{y}) \\
& \Omega_1 = z_1(\boldsymbol{x},\boldsymbol{y}) - z_1^N \\
& \Omega_2 = z_2^N - r \\
& z_1(\boldsymbol{x},\boldsymbol{y}) \geq z_1^N \\
& z_2(\boldsymbol{x},\boldsymbol{y}) \leq z_2^N,
\end{aligned}
\tag{4.11}
$$

where Ω_1 and Ω_2 are dummy non-negative variables to capture the value of $z_1(\boldsymbol{x},\boldsymbol{y}) - z_1^N$ and $z_2^N - r$, respectively. Overall, all constraints in Problem (4.11) are either linear or quadratic. Therefore, this formulation is a BQCQP and can be solved using commercial solvers.

4.3.2 Performance of the Approach: A Numerical Example

We now present a numerical example for showing how the Nash bargaining solution approach works over a predefined set of parcels of land. In this example, there exists a total of 50 parcels in which only 15 of them are already under protection, i.e., $|S| = 15$, and the remaining ones can be considered for investment, i.e., $|B| = 35$. Each parcel has an associated return, variance, and size/area. Additionally, we consider a random correlation between -1 and 1 for each pair of parcels. Interested readers may refer to Appendix C2 to see the details of the data simulated for this numerical example.

The initial protection state of this example is shown in Figure 4.5 in which the total return is 16.81 and the total risk is 0.04. We consider two scenarios to illustrate the performance of our proposed approach. The first scenario imposes no constraint on the feasible set, i.e., $F = \{0, 1\}^{50}$. However, the second scenario limits the feasible set by imposing a budget constraint of the form (4.4). We will use the size of parcels for defining the budget constraint in the second scenario. Specifically, in Inequality (4.4), α_j or β_j represents the ratio of the size of parcel j to the total size of unprotected land at the initial state. For example, because the total size of unprotected land at the initial state is 2547.79 and the size of parcel 1 is 453.63 (based on Appendix C2), we have that $\alpha_1 = 0.178$. Similarly, because the size of parcel 10 is 58.99, we have that $\beta_{10} = 0.023$. Note that parcel 1 is an element of B whereas parcel 10 is an element of S. Finally, for defining the constraint, we also set the available budget to 0.1, i.e., $\gamma = 0.1$ in this example.

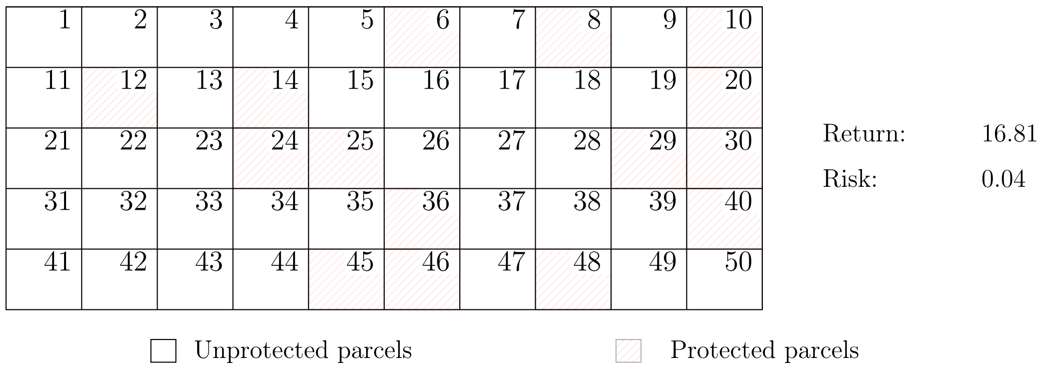


Figure 4.5 – Initial state of conservation for the example

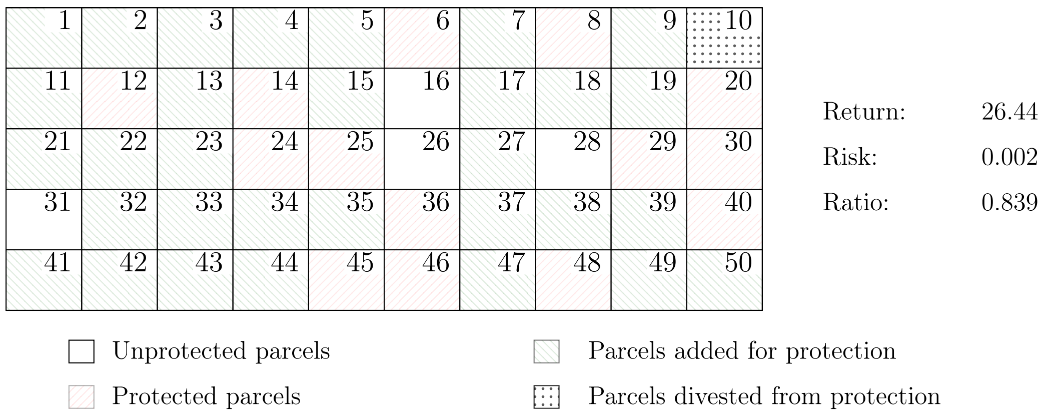


Figure 4.6 – Nash bargaining solution for the example under the first scenario

We first implement the proposed Nash bargaining solution approach for the first scenario. Figure 4.6 shows the obtained solution in which the total return is improved to 26.44 while the risk is reduced to 0.002. In this figure, 'Ratio' shows the ratio of the total size of land added to the reserve to 2547.69 (which is the total size of the unprotected land at the initial state). Intuitively, one can think of this ratio as the value of the mathematical expression on the left-hand-side of Inequality (4.4). The ‘Ratio’ is 0.839 for the Nash bargaining solution of the first scenario. This implies that although parcel 10 is divested in the solution, the total size of land added to the reserve is 83.9% of the total size of the unprotected land at the initial state. To better understand the significant amount of improvement obtained by the Nash bargaining solution, the initial state and the Nash bargaining solution are shown in the criterion space in Figure 4.7. We observe that the obtained Nash bargaining solution significantly outperforms/dominates the initial state.

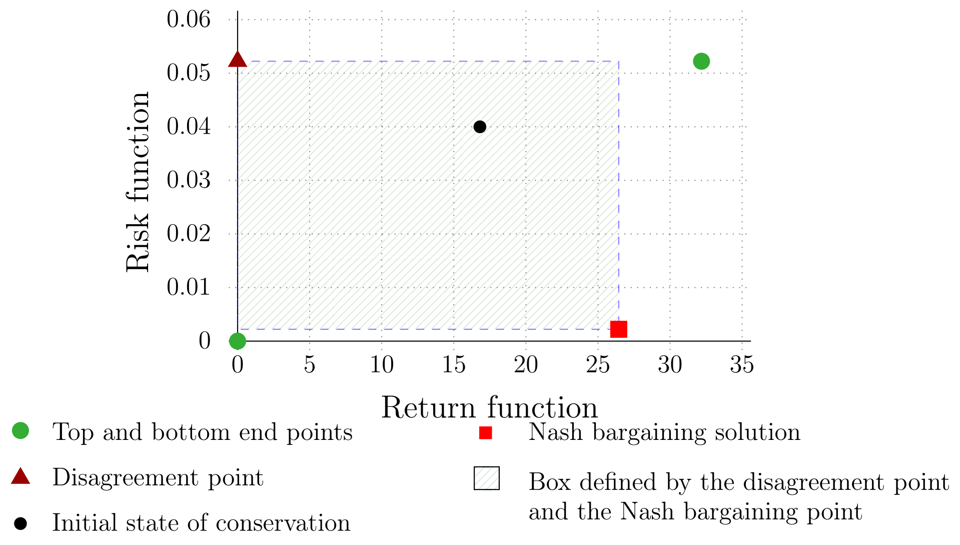


Figure 4.7 – The Nash bargaining solution and the initial state of the example for the first scenario

We now implement the proposed Nash bargaining solution approach for the second scenario. Figure 4.8 shows the obtained solution in which the total return is improved to 26.55 while the risk is reduced to 0.003 (compared to the initial state). The ‘Ratio’ is 0.0996 for the Nash bargaining solution of the second scenario and this makes sense because $\gamma = 0.1$ in the second scenario. This implies that although parcel 10 is divested in the solution, the total size of land added to the reserve is 9.96% of the total size of the unprotected land at the initial state.

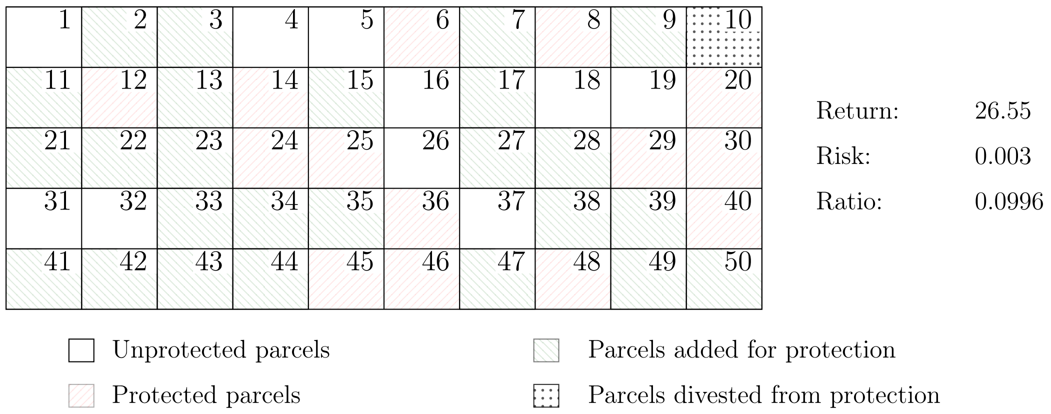


Figure 4.8 – Nash bargaining solution of the example for the second scenario

4.3.3 A Computational Study

In this section, we conduct a comprehensive computational study to show the performance of our proposed Nash bargaining solution approach for MPT-SCP problems. We use the C++ programming language to implement the proposed approach, and employ three different solvers including CPLEX 12.7, Gurobi 8.1 and FICO Xpress 8.5 to solve the optimization problems in our proposed approach. All computational experiments are carried out on a Dell PowerEdge R630 with two Intel Xeon E5-2650 2.2 GHz 12-Core Processors (30MB), 128GB RAM, and the RedHat Enterprise Linux 7.0 operating system. We allow optimization solvers to use up to 10 threads when solving each optimization problem. Also, we impose a time limit of 8 hours for each experiment.

To evaluate the performance of the proposed approach, we randomly generate five classes of instances that are different only in terms of the number of parcels that they have. Specifically, our instances have 50 or 100 or 200 or 400 or 800 parcels. For each class, we randomly generate 5 instances. So, in total, we have 25 instances. We run each instance under 5 different scenarios. The scenarios are generated using the procedure explained in subsection 4.3.2. Specifically, the first scenarios is unconstrained and scenarios 2-5 include a budget constraint based on the size of parcels. The difference between scenarios 2 to 5 is only on the value of γ (see Equation (4.4)). Specifically, γ is set to 0.1, 0.2, 0.35, and 0.5 for scenarios 2-5, respectively.

Our instances are randomly generated based on the `volcano` data set, which is an elevation model in `R` programming language that consists of the topographic information for the volcano Maunga Whau (Mt Eden) in New Zealand (available at `shorturl.at/gDMPZ`). The idea is to simulate spatially heterogeneous species distributions that mimic populations mapping onto natural variation in nature (e.g. elevation), rather than assuming complete spatial randomness in their distribution. In order to explore the overall performance of our approach, we report the following pieces of information,

- *Obj. Value (ratio)*: The ratio of the objective value of the solution obtained by each commercial optimization solver for Problem (4.11) to the objective value of the best solution obtained among the solvers, i.e., CPLEX, Gurobi, and Xpress. Hence, larger values are better and ideally should be equal to one.

- *Optimality Gap*: Since we impose a time limit for our experiments, it is possible that optimization solvers do not find an optimal solution within the time limit. However, they can provide a certificate in percentage about how far they obtained solution is possibly from an optimal solution (in terms of the objective value). This certificate is called the optimality gap and ideally it should be zero. For example, an optimality gap of 10% for a given solver shows that the objective value of Problem (4.11) for the solution obtained by the solver is guaranteed not to be 10% smaller/worse than its optimal objective value.

- Time (seconds): This is the actual solution time (in seconds) for an optimization solver to solve an instance. Note that although we are imposing a time limit of 8 hours (28,800), some instances can be solved in a shorter period of time.

The results for the performance of our approach over the smallest class of instances, i.e., instances with a landscape of 50 parcels, are shown in Table 4.1. The numbers are averages over five instances in all the tables in this section. Also, the best values are bolded in the tables (whenever appropriate). We observe from Table 4.1 that CPLEX outperforms all other solvers. All instances with 50 parcels are solved to optimality in around one second by CPLEX. Both Gurobi and Xpress performed poorly and could not solve many of the instances to optimality within the time limit. This is highlighted by the fact that the average optimality gap of Gurobi and Xpress are 1.53% and 0.04%, respectively. Moreover, there is an increasing tendency in the solution time of Gurobi and Xpress as the budget available, i.e., λ, increases.

Table 4.1 – Overall performance of the proposed approach on the class of instances with 50 parcels

Size	**Scenario**	**Obj. Value (ratio)**			**Optimality Gap**			**Time (seconds)**		
		CPLEX	Gurobi	Xpress	CPLEX	Gurobi	Xpress	CPLEX	Gurobi	Xpress
50	No Budget	**1.00**	0.95	0.94	**0.00%**	6.42%	**0.00%**	**0.51**	12,348.11	5,909.81
	$\lambda = 0.1$	**1.00**	**1.00**	0.75	**0.00%**	**0.00%**	0.02%	0.19	**0.13**	1.00
	$\lambda = 0.2$	**1.00**	**1.00**	**1.00**	**0.00%**	**0.00%**	0.04%	**0.58**	11.97	1.00
	$\lambda = 0.35$	**1.00**	**1.00**	0.87	**0.00%**	**0.00%**	0.11%	**0.86**	982.14	31.40
	$\lambda = 0.5$	**1.00**	**1.00**	0.83	**0.00%**	1.24%	0.05%	**1.04**	13,421.78	4,192.40
Average		**1.00**	0.99	0.88	**0.00%**	1.53%	0.04%	**0.63**	5,352.83	2,027.12

Table 4.2 – Overall performance of the proposed approach on the class of instances with 100 parcels

Size	**Scenario**	**Obj. Value (ratio)**		**Optimality Gap**		**Time (seconds)**	
		CPLEX	Gurobi	CPLEX	Gurobi	CPLEX	Gurobi
100	No Budget	**1.00**	0.97	**0.01%**	26.18%	**1.99**	23,040.01
	$\lambda = 0.1$	**1.00**	**1.00**	**0.00%**	**0.00%**	**1.01**	1,139.76
	$\lambda = 0.2$	**1.00**	**1.00**	**0.00%**	13.28%	**1.46**	21,167.41
	$\lambda = 0.35$	**1.00**	0.99	**0.00%**	18.09%	**2.39**	23,040.01
	$\lambda = 0.5$	**1.00**	0.98	**0.00%**	21.24%	**8.15**	23,040.01
Average		**1.00**	0.99	**0.00%**	15.76%	**3.00**	18,285.44

Unfortunately, for classes of instances with more than 50 parcels, we were unable to obtain any solution using Xpress because of the way that the solver handles the computer memory. We observed a similar issue when using Gurobi but only for classes of instances with more than 100 parcels. So, for the second classes of instances (which has 100 parcels), we only report the results of CPLEX and Gurobi. Our results for the second class can be found in Table 4.2. Again, we observe that CPLEX was able to solve all instances to optimality within 10 seconds. However, Gurobi was not able to solve many instances to optimality and has the optimality gap of around 15.76% on average.

Table 4.3 – Overall performance of the proposed approach on the classes of instances with 200, 400, and 800 parcels

Size	**Scenario**	**Optimality Gap**	**Time (seconds)**
200	No Budget	0.00%	4.71
	$\lambda = 0.1$	0.00%	1.32
	$\lambda = 0.2$	0.00%	2.26
	$\lambda = 0.35$	0.00%	3.17
	$\lambda = 0.5$	0.00%	3.64
400	No Budget	0.18%	23,054.21
	$\lambda = 0.1$	0.08%	6,150.20
	$\lambda = 0.2$	0.04%	14,909.01
	$\lambda = 0.35$	0.16%	17,403.79
	$\lambda = 0.5$	0.18%	17,987.62
800	No Budget	0.06%	23,047.38
	$\lambda = 0.1$	0.00%	19.53
	$\lambda = 0.2$	0.03%	8,051.47
	$\lambda = 0.35$	0.04%	13,327.63
	$\lambda = 0.5$	0.11%	22,476.05

Finally, our results for the remaining classes of instances can be found in Table 4.3. As mentioned earlier, both Gurobi and Xpress were not able to solve such instances and hence their results are not reported. We again observe that CPLEX has performed very well. For class of instances with 200 parcels, it was able to find optimal solutions within a few seconds. For classes of instances with 400 and 800 parcels, it was not able to solve many instances to optimality. However, it was able to find near optimal solutions for even such large instances because the optimality gap is less 0.2% which is quite small.

4.4 Conclusion

We have developed a new approach that directly computes a desirable solution from the Pareto-optimal frontier of large MPT-SCP problems. Our approach is based on the concept of Nash bargaining solution which was first presented by [3], but it expands from this original study in several important ways. Firstly, it overcomes the need to compute the entire Pareto-optimal frontier. Secondly, we showed that we can solve large optimization problems by simply implementing three smaller optimization problems that are either binary linear programs or BQCQPs. Thirdly, we have defined the problem in such a way that it can be solved using existing commercial solvers which should make the implementation of these algorithms more accessible to users who are not experts with optimization algorithms. Specifically, we showed among the commercial solvers CPLEX aligns significantly better with our approach. The numerical results showed that CPLEX can solve instances with 200 parcels to optimality in a few seconds and can find almost optimal solutions for instances with even 800 parcels within 8 hours.

Chapter 5: Learning to Project in Multi-Objective Binary Linear Programming

In this chapter, we present the fourth (final) contribution of this thesis and the only contribution that is not related to the problem of optimization over the efficient set. We present a machine-learning-based solution approach to learn how to project in multi-objective binary linear programming. We first present our motivation behind this study and some background in exact criterion space search algorithms for multi-objective binary linear problems and how do they project on a criterion space with a lower dimension. Second, we show some preliminaries about one of the new effective criterion space search algorithms, the so-called KSA. Third, we present our machine learning framework with a detailed explanation of its characteristics. Finally, we present some concluding remarks.

5.1 Introduction

The development of this study is focused on Multi-Objective Binary Linear Programs (MOBLPs), i.e., multi-objective optimization problems in which all decision variables are binary and all objective functions and constraints are linear. In the last few years, significant advances have been made in the development of effective algorithms for solving MOBLPs (see for instance [13, 1, 14, 15, 4, 5, 35, 6, 7, 8, 9, 10, 11, 12, 36]). Many of the recently developed algorithms fall into the category of criterion space search algorithms, i.e., those that work in the space of objective functions' values. Hence, such algorithms are specifically designed to find all *nondominated points* of a multi-objective optimization problem, i.e., the image of an efficient solution in the criterion space is being referred to as a nondominated point. After computing each nondominated point, criterion space search algorithms remove the proportion of the criterion space dominated by the obtained nondominated point and search for not-yet-found nondominated points in the remaining space.

In general, to solve a multi-objective optimization problem, criterion space search algorithms solve a sequence of single-objective optimization problems. Specifically, when solving a problem with p objective functions, many criterion space search algorithms first attempt to transform the problem into a sequence of problems with $(p-1)$ objectives (Boland et al. [15]). In other words, they attempt to compute all nondominated points by discovering their projections in a $(p-1)$-dimensional criterion space. Evidently, the same process can be applied recursively until a sequence of single-objective optimization problems are generated. For example, to solve each problem with $(p-1)$ objectives, a sequence of problems with $(p-2)$ objectives can be solved.

Overall, there are at least two possible ways to apply the projection from a higher dimensional criterion space (for example p) to a criterion space with one less dimension (for example $p-1$):

- *Weighted Sum Projection*: A typical approach used in the literature (see for instance Özlen and Azizoğlu [83]) is to select one of the objective functions available in the higher dimension (for example $z_1(\boldsymbol{x})$) and remove it after adding it with some strictly positive weight to the other objective functions. In this case, by imposing different bounds for $z_1(\boldsymbol{x})$ and/or the value of the other objective functions, a sequence of optimization problems with $p-1$ objectives will be generated.

- *Lexicographical Projection*: We first note that a *lexicographical optimization problem* is a two-stage optimization problem that attempts to optimize a set of objectives, the so-called *secondary* objectives, over the set of solutions that are optimal for another objective, the so-called *primary* objective. The first stage in the lexicographical optimization problem is a single-objective optimization problem as it optimizes the primary goal. The second stage, however, can be a multi-objective optimization problem as it optimizes the secondary objectives. Based on this definition, another typical approach (see for instance [9]) for projection is to select one of the objective functions available in the higher dimension (for example $z_1(\boldsymbol{x})$) and simply remove it. In this case, by imposing different bounds for $z_1(\boldsymbol{x})$ and/or the value of the other objective functions, a sequence of lexicographical optimization problems should be solved in which $z_1(x)$ is the primary objective and the remaining $p-1$ objectives are secondary objectives.

In light of the above, which objective function should be selected for doing a projection and how to do a projection are two typical questions that can be asked when developing a criterion space search algorithm. So, by this observation, there are many possible ways to develop a criterion space search algorithm and some of which may perform better for some instances. So, the underlying research question of this study is that whether Machine Learning (ML) techniques can help us answer the above questions for a given class of instances of a multi-objective objective optimization problem?

It is worth mentioning that, in recent years, similar questions have been asked in the field of single-objective optimization. For example, ML techniques have been successfully implemented for the purpose of *variable selection* and *node selection* in branch-and-bound algorithms (see for instance [84, 85, 86, 87, 88]). However, still the majority of the algorithmic/theoretical studies in the field of ML have been focused on using optimization models and algorithms to enhance ML techniques and not the other way around (see for instance [89, 90, 91, 92, 93, 94]).

It is evident that if one can show that ML is even valuable for such a high-level question then deeper questions can be asked and explored that can possibly improve the solution time significantly. In order to answer the above question, we employ one of the effective state-of-the-art algorithms in the literature of multi-objective optimization, the so-called KSA which is developed by Kirlik and Sayın [6]. This algorithm uses the lexicographical projection for reducing the p-dimensional criterion space to $p-1$, and then it recursively reduces the dimension from $p-1$ to 1 by using a special case of the weighted sum projection in which all the weights are equal to one.

Currently, the default objective function for conducting the projection from p to $(p-1)$-dimensional criterion space is the first objective function (or better say random because one can change the order of the objective functions in an input file). So, a natural question is that does it really matter which objective function is selected for such a projection? To answer this question, we conducted a set of experiments by using a C++ implementation of the KSA which is publicly available in `http://home.ku.edu.tr/~moolibrary` and recorded the number of ILPs solved (#ILPs) and the computational time (in seconds). We generated 1000 instances (200 per class) of tri-objective Assignment Problem (AP) and 1000 instances (200 per class) of Knapsack Problem (KP) based on the procedure described by Kirlik and Sayın [6]. Table 5.1 shows the impact of projecting based on the worst and best objective function using the KSA where #ILPs is the number of single-objective integer linear programs solved. Numbers reported in this table are averages over 200 instances.

Table 5.1 – Projecting based on different objectives using the KSA.

Type	**#Objectives**	**#Variables**	**Projecting worst objective**		**Projecting best objective**		**%Decrease**	
			Run time (s.)	**#ILPs**	**Run time (s.)**	**#ILPs**	**Run time (s.)**	**#ILPs**
AP	3	20×20	351.56	5,122.67	337.03	5,015.39	4.31%	4.61%
		25×25	948.21	9,685.69	912.07	9,500.52	3.96%	4.40%
		30×30	2,064.34	16,294.37	1,988.64	16,019.64	3.81%	4.01%
		35×35	4,212.69	26,161.34	4,050.44	25,592.13	4.01%	4.27%
		40×40	6,888.45	35,737.21	6,636.79	35,061.41	3.79%	3.97%
KP	3	60	270.34	3,883.45	212.41	3,861.68	27.27%	1.05%
		70	813.31	6,182.04	638.87	6,158.20	27.31%	0.82%
		80	1,740.80	9,297.96	1,375.58	9,265.09	26.55%	0.74%
		90	5,109.56	14,257.32	3,917.32	14,212.35	30.44%	0.63%
		100	10,451.97	19,420.06	7,780.96	19,366.88	34.33%	0.57%

We observe that, on average, the running time can be reduced up to 34% while the #ILPs can be improved up to 4%. This numerical study clearly shows the importance of projection in the solution time. Hence, it is certainly worth studying ML techniques in predicting the best objective function for projecting, the so-called *learning to project*. So, our main contribution in this study is to introduce an ML framework to simulate the selection of the best objective function to project. We collect data from each objective function and their interactions with the decision space to create features. Based on the created features, an easy-to-evaluate function is learned to emulate the classification of the projections. Another contribution of this study is developing a simple but effective bi-objective optimization-based heuristic approach to select the best subset of features to overcome the issue of overfitting. We show that the accuracy of the proposed prediction model can reach up to around 72%, which represents up to 12% improvement in solution time.

5.2 Preliminaries

A *Multi-Objective Binary Linear Program* (MOBLP) is a problem of the form:

$$\min_{\boldsymbol{x} \in \mathcal{X}} \{z_1(\boldsymbol{x}), \dots, z_p(\boldsymbol{x})\}, \tag{5.1}$$

in which $\mathcal{X} := \big\{\boldsymbol{x} \in \{0,1\}^n : A\boldsymbol{x} \leq \boldsymbol{b}\big\}$ represents the *feasible set in the decision space*, $A \in \mathbb{R}^{m\times n}$, and $\boldsymbol{b} \in \mathbb{R}^m$. It is assumed that $\mathcal{X}$ is *bounded* and $z_i(\boldsymbol{x}) = \boldsymbol{c}_i^\intercal \boldsymbol{x}$ where $\boldsymbol{c}_i \in \mathbb{R}^n$ for $i = 1, 2, \dots, p$ represents a linear objective function. The image $\mathcal{Y}$ of $\mathcal{X}$ under vector-valued function $\boldsymbol{z} := (z_1, z_2, \dots, z_p)^\intercal$ represents the *feasible set in the objective/criterion space*, that is $\mathcal{Y} := \{\boldsymbol{o} \in \mathbb{R}^p : \boldsymbol{o} = \boldsymbol{z}(\boldsymbol{x}) \text{ for all } \boldsymbol{x} \in \mathcal{X}\}$.

Overall, multi-objective optimization is concerned with finding all nondominated points, i.e., an exact representation of the elements of $\mathcal{Y}_N$. The set of nondominated points of a MOBLPs is finite (since by assumption $\mathcal{X}$ is bounded). However, due to the existence of *unsupported* nondominated points, i.e., those nondominated points that cannot be obtained by optimizing any positive weighted summation of the objective functions over the feasible set, computing all nondominated points is challenging. One of the effective criterion space search algorithms for MOBLPs is the KSA and its high-level description is provided next.

The KSA is basically a variation of the ε-constraint method for generating the entire nondominated frontier of multi-objective integer linear programs. In each iteration, this algorithm solves the following lexicographical optimization problem in which the first stage is:

$$\hat{\boldsymbol{x}} \in \arg\min \big\{ z_1(\boldsymbol{x}) : \boldsymbol{x} \in \mathcal{X},\ z_i(\boldsymbol{x}) \le u_i \ \ \forall i \in \{2, \dots, p\} \big\},$$

where $u_2, \dots, u_p$ are user-defined upper bounds. If $\hat{\boldsymbol{x}}$ exists, i.e., the first stage is feasible, then the following second-stage problem will be solved:

$$\hat{\boldsymbol{x}}^* \in \arg\min \big\{ \sum_{i=2}^{p} z_i(\boldsymbol{x}) : \boldsymbol{x} \in \mathcal{X},\ z_1(\boldsymbol{x}) \le z_1(\hat{\boldsymbol{x}}),\ z_i(\boldsymbol{x}) \le u_i \ \ \forall i \in \{2, \dots, p\} \big\}.$$

The algorithm computes all nondominated points by imposing different values on $u_2, \dots, u_p$ in each iteration. Interested readers may refer to Kirlik and Sayın [6] for further details about how values of $u_2, \dots, u_p$ will be updated in each iteration. It is important to note that in the first stage users can replace the objective function $z_1(\boldsymbol{x})$ with any other arbitrary objective function, i.e., $z_j(\boldsymbol{x})$ where $j \in \{1, \dots .p\}$, and change the objective function of the second stage accordingly, i.e., $\sum_{i=1:i \neq j}^{p} z_i(\boldsymbol{x})$. As shown in Introduction, on average, the running time can decrease up to 34% by choosing the right objective function for the first stage. So, the goal of the proposed machine learning technique in this study is to identify the best choice.

As an aside, we note that to be consistent with our explanation of the lexicographic and/or weighted sum projections in Introduction, the lexicographic optimization problem of the KSA is presented slightly differently in this section. Specifically, Kirlik and Sayın [6] use the following optimization problem instead of the second-stage problem (mentioned above):

$$\hat{\boldsymbol{x}}^* \in \arg\min \big\{ \sum_{i=1}^{p} z_i(\boldsymbol{x}) : \boldsymbol{x} \in \mathcal{X},\ z_1(\boldsymbol{x}) = z_1(\hat{\boldsymbol{x}}),\ z_i(\boldsymbol{x}) \le u_i \ \ \forall i \in \{2, \dots, p\} \big\}.$$

However, one can easily observe that these two formulations are equivalent. In other words, the lexicographic optimization problem introduced in this section is a just different representation of the one proposed by Kirlik and Sayın [6].

5.3 Machine Learning Framework

We now introduce our ML framework for learning to project in MOBLPs. Our proposed framework is based on Multi-class Support Vector Machine (MSVM). In this application, MSVM learns a function $f : \Phi \to \Omega$ from a training set to predict which objective function will have the best performance in the first stage of the KSA (for a MOBLP instance), where Φ is the feature map domain describing the MOBLP instance and $\Omega := \{1, 2, \dots, p\}$ is the domain of the labels. A label $y \in \Omega$ indicates the index of the objective function that should be selected. We do not explain MSVM in this study but interested readers may refer to Crammer and Singer [95] and Tsochantaridis et al. [96] for details. We used the publicly available implementation of MSVM in this study which can be found in `https://goo.gl/4hLjyq`. It is worth mentioning that we have used MSVM mainly because it was performing well during the course of this study. In subsection 5.3.4.4, we also report results obtained by replacing MSVM with Random Forest (Breiman [97], Prinzie and Van den Poel [98]) to show the performance of another learning technique in our proposed framework. Also, we provide more reasons in subsection 5.3.4.4 about why MSVM is used in this study. Overall, the proposed ML framework contains three main components:

- *Component 1*: It is evident that by changing the order of the objective functions of a MOBLP instance in an input file, the instance remains the same. Therefore, in order to increase the stability of the prediction of MSVM, we propose an approach to pre-order the objective functions of each MOBLP instance in an input file before feeding it to MSVM (see subsection 5.3.1).

- *Component 2*: We propose several generic features that can be used to describe each MOBLP instance. A high-level description of the features can be found in subsection 5.3.2.1 and their detailed descriptions can be found in Appendix D1.

- *Component 3*: We propose a bi-objective heuristic approach (see subsection 5.3.3) for selecting the best subset of features for each class of MOBLP instances (which are AP and KP in this study). Our numerical results show that our approach selects around 15% of features based on the training set for each class of MOBLP instances in practice. Note that identifying the best subset of features is helpful for overcoming the issue of overfitting and improving the prediction accuracy (Charkhgard and Eshragh [99], Tibshirani [100]).

The proposed ML framework uses the above components for training purposes. A detailed discussion on the accuracy of the proposed framework on a testing set (for each class of MOBLP instances) is given in subsection 5.3.4.

5.3.1 A Pre-Ordering Approach for Objective Functions

It is obvious that by changing the order of objective functions in an input file corresponding to an instance, a new instance will not be generated. In other words, only the instance is represented differently in that case and hence its nondominated frontier will remain the same. This suggests that the vector of features that will be extracted for any instance should be independent of the order of the objective functions. To address this issue, we propose to perform a pre-ordering (heuristic) approach before giving an instance to MSVM for training or testing purposes. That is, when users provide an instance, we first change its input file by re-ordering the objective functions before feeding it to the MSVM. Obviously, this somehow stabilizes the prediction accuracy of the proposed ML framework.

In light of the above, let

$$\tilde{\boldsymbol{x}} := (\frac{1}{\sum_{i=1}^{p}|c_{i1}|+1}, \dots, \frac{1}{\sum_{i=1}^{p}|c_{in}|+1}).$$

In the proposed approach, we re-order the objective functions in an input file in a non-decreasing order of $\boldsymbol{c}_1^\intercal\tilde{\boldsymbol{x}}, \boldsymbol{c}_2^\intercal\tilde{\boldsymbol{x}}, \ldots, \boldsymbol{c}_p^\intercal\tilde{\boldsymbol{x}}$. Intuitively, $\boldsymbol{c}_i^\intercal\tilde{\boldsymbol{x}}$ can be viewed as the normalization score for objective function $i \in \{1, \ldots, p\}$. In the rest of this chapter, the vector $\boldsymbol{c}_i$ for $i = 1, 2, \ldots, p$ is assumed to be ordered according to the proposed approach.

5.3.2 Features and Labels Describing a MOBLP Instance

This section provides a high-level explanation of the features that we create to describe a MOBLP instance and also how each instance is labeled. To the best of our knowledge, there are no studies that introduce features to describe multi-objective instances, and hence the proposed features are new.

5.3.2.1 Features

The efficiency of our proposed ML approach relies on the features describing a MOBLP instance. In other words, the features should be easy to compute and effective. Based on this observation, we create only *static* features, i.e., those that are computed once using just the information provided by the MOBLP instance. Note that we only consider static features because the learning process and the decision on which objective function to select for projection (in the KSA) have to take place before solving the MOBLP instance. Overall, due to the nature of our research, most of our features describe the objective functions of the instances. We understand that the objective functions by themselves are a limited source of information to describe an instance. Therefore, we also consider establishing some relationships between the objective functions and the other characteristics of the instance in order to improve the reliability of our features.

In light of the above, a total of $5p^2 + 106p - 50$ features are introduced for describing each instance of MOBLP. As an aside, because in our computational study $p = 3$, we have 313 features in total. Some of these features rely on the characteristics of the objective functions such as the magnitude and the number of positive, negative and zero coefficients.

We also consider features that establish a relationship between the objective functions using some normalization techniques, e.g., the pre-ordering approach used to order the objective functions (see subsection 5.3.1). Other features are created based on some mathematical and statistical computations that link the objective functions with the technological coefficients and the right-hand-side values.

We also define features based on the area of the projected criterion space i.e., the corresponding $(p-1)$-dimensional criterion space, that needs to be explored when one of the objectives is selected for conducting the projection. Note that, to compute such an area, several single-objective binary linear programming problems need to be solved. However, in order to reduce the complexity of the features extraction, we compute an approximation of the area-to-explore by optimizing the linear relaxation of the problems. Additionally, we create features in which the basic characteristics of an instance are described, e.g., size, number of variables, and number of constraints.

The main idea of the features is to generate as much information as possible in a simple way. We accomplish this by computing all the proposed features in polynomial time for a given instance. The features are normalized using a t-statistic score. Normalization is performed by aggregating subsets of features computed from a similar source. Finally, the values of the normalized feature matrix are distributed approximately between -1 and 1. Interested readers can find a detailed explanation of the features in Appendix D1.

5.3.2.2 Labels

Based on our research goal, i.e., simulating the selection of the best objective, we propose a multi-class integer labeling scheme for each instance, where $y \in \Omega$ is the label of the instance and $\Omega = \{1, 2, \ldots, p\}$ is the domain of the labels. The value of y^i classifies the instance i with a label that indicates the index of the best objective function for projection based on the running time of the KSA (when generating the entire nondominated frontier of the instance). The label of each instance is assigned as follows:

$$y^i \in \arg\min_{j\in\{1,\ldots,p\}} \{\text{RunningTime}^i_j\}, \tag{5.2}$$

where RunningTime^i_j is the running time of the instance i when the objective function j is used for projecting the criterion space.

5.3.3 Best Subset Selection of Features

It is easy to observe that by introducing more (linearly independent) features and re-training an ML model (to optimality) its prediction error, i.e., $error = 1 - accuracy$, on the training set decreases and eventually it becomes zero. This is because in that case we are providing a larger degree of freedom to the ML model. However, this is not necessarily the case for the testing set. In other words, by introducing more features, the ML model that will be obtained is often overfitted to the training set and does not perform well on the testing set. So, the underlying idea of the best subset selection of features is to avoid the issue of overfitting. However, the key point is that in a real-world scenario we do not have access to the testing set. So, selecting the best subset of features should be done based the information obtained from the training set.

In light of the above, studying the trade-off between the number of features and the prediction error of an ML model on the training set is helpful for selecting the best subset of features (Charkhgard and Eshragh [99]). However, computing such a tradeoff using exact methods is difficult in practice since the total number of subsets (of features) is an exponential function of the number of features. Therefore, in this section, we introduce a bi-objective optimization-based heuristic for selecting the best subset of features. The proposed approach has two phases:

- *Phase I*: In the first phase, the algorithm attempts to approximate the tradeoff. Specifically, the algorithm computes an approximated nondominated frontier of a bi-objective optimization problem in which its conflicting objectives are minimizing the number of features and minimizing the prediction error on the training set.

- *Phase II*: In the second phase, the algorithm selects one of the approximated nondominated point and its corresponding MSVM model to be used for prediction on the testing set.

We first explain Phase I. To compute the approximated nondominated frontier, we run MSVM iteratively on a training set. In each iteration, one approximated nondominated point will be generated. The approximated nondominated point obtained in iteration t is denoted by (k_t, e_t) where k_t is the number of features in the corresponding prediction model (obtained by MSVM) and e_t is the prediction error of the corresponding model on the training set.

To compute the first nondominated point, the proposed approach/algorithm assumes that all features are available and it runs the MVSM to obtain the parameters of the prediction model. We denote by W^t the parameter of the prediction model obtained by MSVM in iteration t. Note that W^t is a $p \times k_t$ matrix where p is the number of objectives. Now consider an arbitrary iteration t. The algorithm will explore the parameters of the prediction model obtained in the previous iteration by MSVM, i.e., W^{t-1}, and will remove the least important feature based on W^{t-1}. Hence, because of removing one feature, we have that $k_t = k_{t-1} - 1$. Specifically, each column of matrix W^{t-1} is associated to a feature. Therefore, the algorithm computes the standard deviation of each column independently. The feature with the minimum standard deviation will be selected and removed in iteration t. Note that MSVM creates a model for each objective function and that is the reason that matrix W^{t-1} has p rows. So, if the standard deviation of a column in the matrix is zero then we know that the corresponding feature is contributing exactly the same in all p models and therefore it can be removed. So, we observe that the standard deviation plays an important role in identifying the least important feature.

Overall, after removing the least important feature, MSVM should be run again for computing W^t and e_t. The algorithm for computing the approximated nondominated frontier terminates as soon as $k^t = 0$. A detailed description of the algorithm for computing the approximated nondominated frontier can be found in Algorithm 3.

Algorithm 3: Phase I: Computing an approximated nondominated frontier

```
   input: Training set, The set of features
 1 $Queue.create(Q)$
 2 $t \leftarrow 1$
 3 $k_t \leftarrow$ The initial number of features
 4 while $k_t \neq 0$ do
 5     if $t \geq 1$ then
 6         Find the least important feature from $W^{t-1}$ and remove it from the set of features
 7         $k_t \leftarrow k_{t-1} - 1$
 8     end
 9     Compute $W^t$ by applying MSVM on the training set using the current set of features
10     Compute $e_t$ by applying the obtained prediction model associated with $W^t$ on the
         training set
11     $Q.add\big((k_t, e_t)\big)$
12     $t \leftarrow t + 1$
13 end
14 return $Q$
```

In the second phase, we select an approximated nondominated point. However, before doing so, it is worth mentioning that MSVM can take a long time to compute W^t in each iteration of Algorithm 3. So, to avoid this issue, users usually terminate MSVM before it reaches to an optimal solution by imposing some termination conditions including a relative optimality gap tolerance and adjusting the so-called *regularization* parameter (see Crammer and Singer [95] and Tsochantaridis et al. [96] for details). In this study, we set the tolerance to 0.1 and the regularization parameter to 5×10^4 (since we numerically observed that MSVM performs better in this case). Such limitations obviously impact the prediction error that will be obtained on the training set, i.e., e^t. So, some of the points that will be reported by Algorithm 3 may dominate each other. Therefore, in Phase II, we first remove the dominated points. In the remaining of this section we assume that there is no dominated point in the approximated nondominated frontier.

Next, the proposed approach selects an approximated nondominated point that has the minimum Euclidean distance from the (imaginary) *ideal* point, i.e., an imaginary point in the criterion space that has the minimum number of features as well as the minimum prediction error. Such a technique is a special case of optimization over the frontier. We note that in bi-objective optimization, the ideal point can be computed easily based on the endpoints of the (approximated) nondominated frontier. Let (k', e') and (k'', e'') be the two endpoints in which $k' < k''$ and $e' > e''$. In this case, the ideal point is (k', e''). Note too that because the first and second objectives have different scales, in this study, we first normalize all approximated nondominated points before selecting a point. Let (k, e) be an arbitrary approximated nondominated point. After normalization, this point will be as follows:

$$(\frac{k-k'}{k''-k'}, \frac{e-e''}{e'-e''}).$$

Observe that the proposed normalization technique ensures that the value of each component of a point will be between 0 and 1. As a consequence, in this case, the nominalized ideal point will be always $(0, 0)$. We will discuss about the effectiveness of our proposed best subset selection approach in the next section.

5.3.4 A Computational Study

In this section, we conduct an extensive computational study to evaluate the performance of the KSA when the proposed ML technique is used for learning to project. We generate 1000 tri-objective AP instances and also 1000 tri-objective KP instances based on the procedures described by Kirlik and Sayın [6]. Since there are three objectives, we compute the entire representation of the nondominated frontier three times for each instance using the KSA; In each time, a different objective function will be selected for projection. We employ CPLEX 12.7 as the single-objective binary linear programming solver. All computational experiments are carried out on a Dell PowerEdge R630 with two Intel Xeon E5-2650

2.2 GHz 12-Core Processors (30MB), 128GB RAM, and the RedHat Enterprise Linux 6.8 operating system. We only use a single thread for all experiments.

Our experiments are divided into three parts. In the first part, we run our approach over the entire set of instances using 80% of the data as the training set and 20% of the data as the testing set. The second part evaluates the prediction models obtained in the first part on a reduced testing set. In other words, the training set is as same as the one in the first part but the testing set is smaller. Specifically, if it does not really matter which objective function to be selected for projection (in terms of solution time) for an instance in the testing set of the first part then we remove it. Obviously one can think of such instances as tie cases. In the third part of our experiments, we extend the concept of reduced testing set to the training set. That is, we remove not only the tie cases from the testing set but also from the training set. In general, the goal of reducing testing set and/or training set is to improve the overall accuracy of the prediction model.

At the end of the computational study, we replace MSVM by Random Forest in the proposed ML framework to show the performance of another learning technique. We note that in this computational study, we do not report any time for our proposed ML framework because the aggregated time of generating the features, learning process, and predictions for all 1000 instances of a class of optimization problem, i.e., AP and KP, is around 50 seconds. This implies that on average almost 0.05 seconds are spent on each instance.

5.3.4.1 Complete Training and Testing Sets

The first part of our experiments are done on the entire training and testing sets. For each class of optimization problems, i.e., KP and AP, the proposed subset selection approach is run on its corresponding training set. The proposed approach obtains the best subset of features and its corresponding prediction model for each class of instances. Before providing detailed explanations about the accuracy of such a prediction model on the testing set, it is necessary to show that the proposed bi-objective optimization approach for selecting the best subset of features is indeed effective.

Figure 5.1 shows the approximated nondominated frontier (obtained during the course of the proposed best subset selection approach) for each class of optimization problems. In this figure, small (red) plus symbols are the outputs of Algorithm 3. The (black) square on the vertical axis shows the ideal point and finally the (yellow) square on the approximated nondominated frontier shows the selected point by the proposed method. First note that we introduced 313 generic features in this study but the tail of the approximated nondominated frontier in Figure 5.1 clearly shows that not all 313 features are used. This is because some of the 313 features are not applicable to all classes and will be removed automatically before running the proposed best subset selection approach.

We observe from Figure 5.1 that, overall, by introducing more features the prediction error deceases for the training set. It is evident when all features are included the accuracy, i.e., $1 - error$, for the training set is around 59.5% and 70% for AP and KP instances, respectively. Of course this is not surprising because the learning procedure will be done based on the training set and introducing more features gives a larger degree of freedom to the learning model. However, this is not necessarily the case for the testing set. Basically, by introducing more features, we may raise the issue of *overfitting*, i.e., the prediction error is small for the training set but large for the testing set.

To show this, for each of the points (other than the ideal point) in Figure 5.1, we have plotted its corresponding point for the testing set in Figure 5.2. Specifically, for each point in Figure 5.1, we run its corresponding model on the testing set to compute the error. From Figure 5.2 we observe that the error highly fluctuates. In fact, it is evident that for AP instances, the prediction model that has around 40 features is the best prediction model. Similarly, from Figure 5.2, it is evident that for KP instances, the prediction model that has around 25 features is the best prediction model.

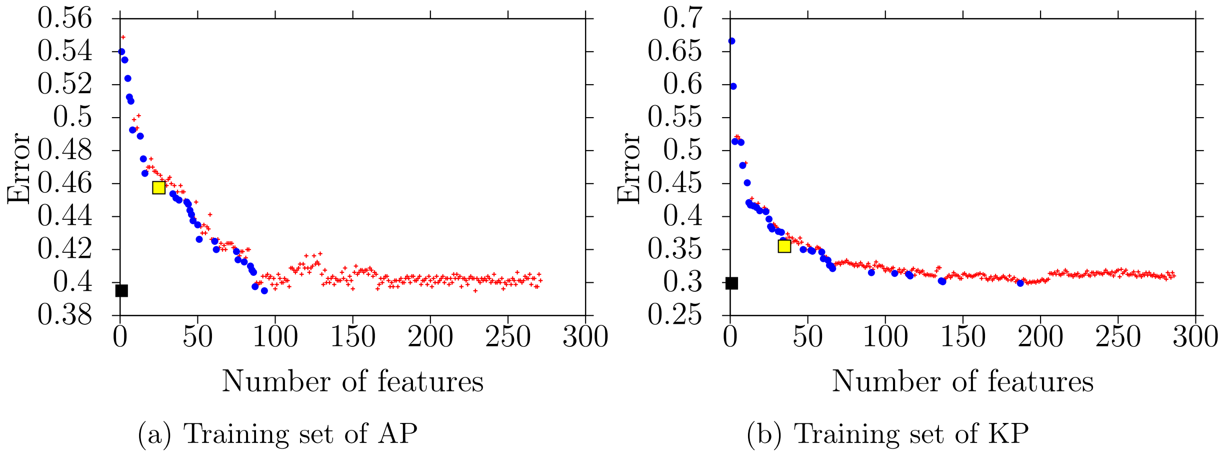
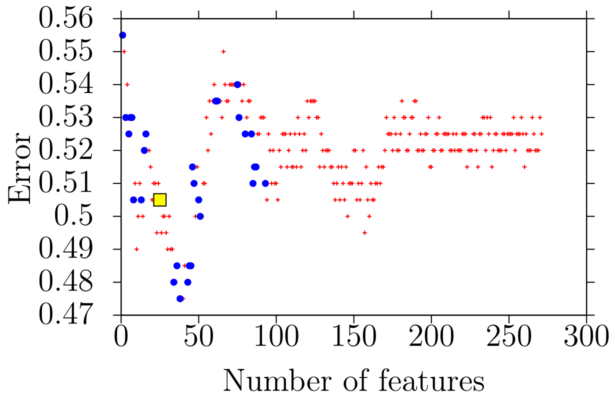


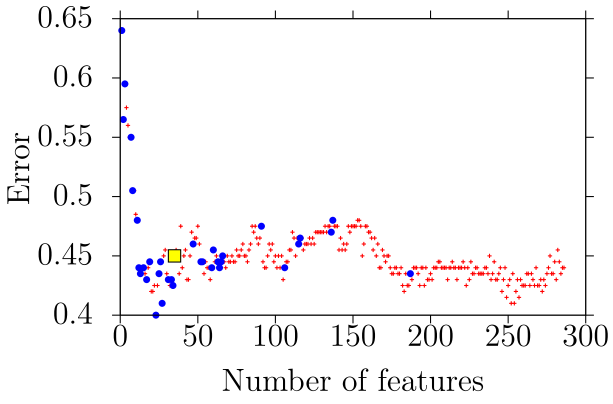
Figure 5.1 – An illustration of the performance of the proposed approach for selecting the best subset of features on the complete training set

Note that in practice, we do not have access to the testing set. So, we should select the best subset of features only based on the training set. Therefore, the goal of any best subset selection technique is to identify the prediction model that is (almost) optimal for the testing set based on the existing data set, i.e., the training set. From Figure 5.2, we observe that our proposed best subset selection heuristic has such a desirable characteristic in practice. We see that the selected model, i.e., the (yellow) square, is nearly optimal. In fact the proposed approach has selected a prediction model with the accuracy of around 50% and 55% for AP and KP instances, respectively. This implies that the absolute difference between the accuracy of the model selected by the proposed subset selection approach and the accuracy of the optimal model is almost 3% and 5% for AP and KP instances, respectively. We note that for both classes of instances less than 50 features exist in the model selected by the proposed approach. Overall, these results are promising due to the fact that the (expected) probability of randomly picking the correct objective function to project is $\frac{1}{p}$, i.e., around 33.3% for the tri-objective instances.

We now discuss about the performance of the selected model in detail for each class of optimization problems. Table 5.2 summarizes our findings. In this table, the column labeled 'Accuracy' shows the average percentage of the prediction accuracy of the selected



(a) Testing set of AP



(b) Testing set of KP

Figure 5.2 – An illustration of the performance of the proposed approach for the best subset selection of features on the complete testing set

model for different subclasses of instances. Note that as mentioned in Introduction, each subclass has 200 instances. The column labeled 'ML vs. Rand' shows the average percentage of decrease in solution time when ML technique is used compared to randomly picking an objective function for projecting. The column labeled 'Best vs. Rand' shows the average percentage of decrease in solution time when the best objective function is selected for projection compared to randomly picking an objective function for projecting. Finally, column labeled '$\frac{\text{ML vs. Rand}}{\text{Best vs. Rand}}$' shows the percentage of 'ML vs. Rand' to 'Best vs. Rand'.

Overall, we observe that our ML method improves the computational time in all testing sets. For AP instances, the improvement is around 0.68% on average which is small. However, we should note that in the ideal case, we could obtain around 1.74% improvement in time on average for such instances. So, the improvement obtained with the proposed ML technique is 38.9% of the ideal case. For largest subclass of AP instances, this number is around 64.99%. For the KP instances, the results are even more promising since the amount of improvement in solution time is around 5.67% on average. In the ideal case, we could obtain an average improvement of 11.17% for such instances. So, the improvement obtained with the proposed ML technique is 50.77% of the ideal case.

Table 5.2 – Accuracy and average time decrease of testing set when using the proposed ML technique (for the case of the complete training and testing sets)

Type	**Vars**	**Accuracy**	**Time Decrease**		$\frac{\textbf{ML vs. Rand}}{\textbf{Best vs. Rand}}$
			ML vs. Rand	**Best vs. Rand**	
AP	20×20	55.56%	1.29%	2.33%	55.43%
	25×25	44.74%	0.67%	1.65%	40.57%
	30×30	41.30%	0.18%	1.48%	12.20%
	35×35	52.08%	0.44%	1.69%	25.81%
	40×40	56.25%	1.07%	1.65%	64.99%
Avg		**49.50%**	**0.68%**	**1.74%**	**38.90%**
KP	60	63.64%	9.12%	11.03%	82.68%
	70	45.65%	2.01%	10.14%	19.79%
	80	56.10%	4.59%	11.42%	40.22%
	90	58.82%	8.05%	13.27%	60.68%
	100	51.43%	5.09%	10.34%	49.24%
Avg		**55.00%**	**5.67%**	**11.17%**	**50.77%**

5.3.4.2 Complete Training Set and Reduced Testing Set

In this section, we test the performance of the model obtained in subsection 5.3.4.1 on a reduced testing set. Specifically, we remove the instances that can be considered as tie cases, i.e., those in which the solution time does not change significantly (relative to other instances) when different objective functions are selected for projection. To reduce the testing set we apply the following steps:

- *Step 1:* We compute the *time range* of each instance, i.e., the difference between the best and worst solution times that can be obtained for the instance when different objective functions are considered for projection.
- *Step 2:* For each subclass of instances, i.e., those with the same number of decision variables, we compute the standard deviation and the minimum of time ranges in that subclass.
- *Step 3:* We eliminate an instance, i.e., consider it as a tie case, if its time range is not greater than the sum of the minimum and standard deviation of time ranges in its associated subclass.

As a result of the procedure explained above, the testing set was reduced by 35.5% for AP instances and by 17.5% for KP instances. Table 5.3 summarizes our findings for the reduced testing set.

Table 5.3 – Accuracy and average time decrease of testing set when using the proposed ML technique (for the case of the complete training set and the reduced testing set)

Type	Vars	Accuracy	Time Decrease		$\frac{\text{ML vs. Rand}}{\text{Best vs. Rand}}$
			ML vs. Rand	**Best vs. Rand**	
AP	20 × 20	58.06%	1.48%	2.54%	58.10%
	25 × 25	60.00%	0.92%	2.23%	41.32%
	30 × 30	40.74%	0.12%	1.96%	6.02%
	35 × 35	57.89%	0.54%	1.97%	27.17%
	40 × 40	69.57%	1.64%	2.15%	76.09%
Avg		**56.59%**	**0.90%**	**2.16%**	**41.73%**
KP	60	71.05%	10.11%	11.96%	84.52%
	70	45.95%	2.03%	11.48%	17.67%
	80	57.58%	5.30%	13.48%	39.31%
	90	62.96%	10.21%	15.57%	65.59%
	100	60.00%	6.32%	11.33%	55.77%
Avg		**59.39%**	**6.66%**	**12.63%**	**52.74%**

Observe that the accuracy of the prediction models has increased significantly for the reduced testing set. Specifically, it has reached to around 56.59% and 59.39% on overage for AP and KP instances, respectively. Since the eliminated instances are considered as tie cases, we can assume that they are also success cases for the prediction model. So, by considering such success cases, the prediction accuracy will increase to $56.59 \times (1 - 0.355) + 35.5 \approxeq 72\%$ and $59.39 \times (1 - 0.175) + 17.5 \approxeq 66.5\%$ for AP and KP instances, respectively. In terms of computational time, we also observe (from Table 5.3) an improvement of around 0.90% and 6.66% on average for AP and KP instances, respectively. This amount of improvement is about 41.73% and 52.74% of the ideal scenarios (on average) for AP and KP instances, respectively.

5.3.4.3 Reduced Training and Testing Sets

Due to promising results obtained in subsection 5.3.4.1, it is natural to ask whether we can see even more improvement if we reduce not only the testing set but also the training set. Therefore, in this section, we eliminate the tie cases using the same procedure discussed in subsection 5.3.4.2 from both training and testing sets. By doing so, the size of the training+testing set was reduced by 37% and 18% for AP and KP instances, respectively.

It is evident that due to the change in the training set, we need to apply our proposed approach for best subset selection of features again. So, similar to subsection 5.3.4.1, Figure 5.3 shows the approximated nondominated frontier for each class of optimization problems based on the reduced training set. By comparing the ideal points in Figures 5.1 and 5.3, an immediate improvement in the (ideal) accuracy can be observed. In fact the absolute difference between the error of the ideal points (in these figures) is around 12% and 7% for AP and KP instances, respectively. Similar improvements can be observed by comparing the selected approximated nondominated points in Figures 5.1 and 5.3.

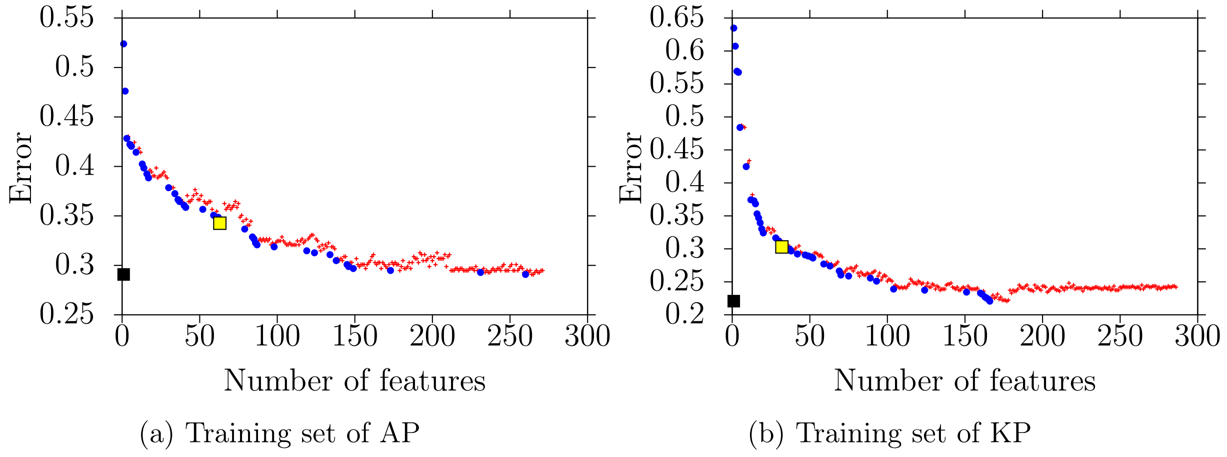
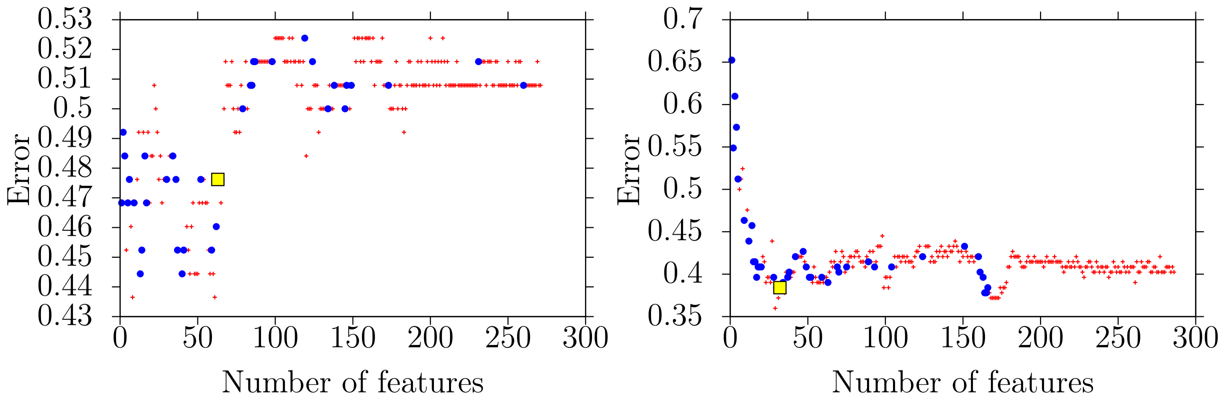


Figure 5.3 – An illustration of the performance of the proposed approach for selecting the best subset of features on the reduced training set

Similar to subsection 5.3.4.1, for each of the points (other than the ideal point) in Figure 5.3, we have plotted its corresponding point for the testing set in Figure 5.4. We again observe that the selected model, i.e., the (yellow) square, is nearly optimal for both classes of optimization problems. In fact the proposed approach has selected a prediction model with the accuracy of around 52% and 62% for AP and KP instances, respectively. This implies that the absolute difference between the accuracy of the model selected by the proposed approach and the accuracy of the optimal model is almost 5% and 3% for AP and KP instances, respectively.



(a) Testing set of AP (b) Testing set of KP

Figure 5.4 – An illustration of the performance of the proposed approach for the best subset selection of features on the reduced testing set

Table 5.4 – Accuracy and average time decrease of testing set when using the proposed ML technique (for the case of the reduced training and testing sets)

Type	Vars	Accuracy	Time Decrease: ML vs. Rand	Time Decrease: Best vs. Rand	ML vs. Rand / Best vs. Rand
AP	20 × 20	58.33%	0.82%	2.14%	38.41%
	25 × 25	52.38%	1.17%	2.39%	48.76%
	30 × 30	55.17%	0.67%	1.93%	34.46%
	35 × 35	45.45%	1.10%	2.53%	43.35%
	40 × 40	52.63%	1.60%	2.87%	55.61%
Avg		**52.38%**	**1.03%**	**2.35%**	**43.99%**
KP	60	60.61%	6.75%	12.16%	55.51%
	70	55.56%	6.79%	12.03%	56.43%
	80	60.00%	12.14%	16.91%	71.79%
	90	72.73%	9.22%	12.26%	75.17%
	100	62.50%	4.70%	10.56%	44.50%
Avg		**61.59%**	**7.61%**	**12.64%**	**60.18%**

A summary of the results of this last experiment can be found in Table 5.4. Observe that the average prediction accuracy on the testing set for the experimental setting in this section, i.e., reduced training and testing sets, has improved significantly for KP instances compared to the results given in subsections 5.3.4.1 and 5.3.4.2. However for the instances of AP problem, the average prediction accuracy in this section is only better than the one presented in subsection 5.3.4.1. Overall, the average prediction accuracy is 52.38% and 61.59% for AP and KP instances when using reduced training and testing sets. By considering the tie cases as success events, the projected accuracy increases up to 70% and 68.5% for AP and KP instances, respectively. The importance of such an increase in the accuracy is highlighted by the time decrease percentages given in Table 5.4, which is over 1% for AP instances and is near 8% for KP instances. In fact, for the largest subclass of AP instances, the average time improvement of 1.6% is equivalent to almost 110 seconds on average. Similarly, for the largest subclass of KP instances, the time improvement of 4.7% is around 490 seconds on average.

5.3.4.4 Replacing MSVM by Random Forest

One main reason that we used MSVM in this study is that (as shown in the previous sections) it performs well in practice for the purpose of this study. However, another critical reason is the fact that MSVM creates a matrix of parameters denoted by W^t in each iteration. This matrix has p rows where p is the number of objective functions. In other words, for each objective function, MSVM creates a specific model for predicting which one should be used for projection in KSA. This characteristic is desirable because it allowed us to develop a custom-built bi-objective heuristic for selecting the best subset of features. Specifically, as discussed in subsection 5.3.3, this characteristic is essential for identifying the least important feature in each iteration of Algorithm 3. However, applying such a procedure on other ML techniques is not trivial.

In light of the above, in this section we replace MSVM by Random Forest within the proposed machine learning framework. However, we simply use the best subset of features selected by MSVM and then feed it to Random Forest for training and predicting. To implement Random Forest we use `scikit-learn` library in Python (Pedregosa et al. [101]). Table 5.5 shows a comparison between the prediction accuracy of MSVM and Random Forest under three experimental settings described in subsections 5.3.4.1-5.3.4.3. In other words, Setting 1 corresponds to the complete training and testing sets; Setting 2 corresponds to the complete training set and reduced testing sets; Finally, Setting 3 refers to the reduced training and testing sets.

In this table, columns labeled 'Increase' show the average percentage of increase in the prediction accuracy of the Random Forest compared to MSVM. Observe from these columns that the reported numbers are mostly negative. This implies that, in general, MSVM outperforms Random Forest in terms of prediction accuracy. For example, in Setting 3, we observe that the accuracy of Random Forest is around 18.63% and 22.88% worse than the accuracy of MSVM on average. This experiment clearly shows the advantage of using MSVM in the proposed ML framework.

Table 5.5 – A performance comparison between MSVM and Random Forest on a testing set

		Setting 1		**Setting 2**		**Setting 3**	
Type	**Vars**	**Accuracy**	**Increase**	**Accuracy**	**Increase**	**Accuracy**	**Increase**
AP	20×20	61.11%	9.99%	64.52%	11.12%	36.36%	-37.66%
	25×25	42.11%	-5.89%	50.00%	-16.67%	26.67%	-49.09%
	30×30	45.65%	10.54%	37.04%	-9.09%	52.63%	-4.60%
	35×35	62.50%	20.01%	65.79%	13.65%	70.59%	55.31%
	40×40	50.00%	-11.11%	73.91%	6.24%	50.00%	-5.00%
Avg		**52.50%**	**6.06%**	**59.69%**	**5.48%**	**42.62%**	**-18.63%**
KP	60	47.73%	-25.00%	47.37%	-33.33%	41.18%	-32.06%
	70	50.00%	9.53%	56.76%	23.52%	50.00%	-10.01%
	80	48.78%	-13.06%	54.55%	-5.27%	61.90%	3.17%
	90	44.12%	-25.00%	55.56%	-11.76%	47.06%	-35.30%
	100	37.14%	-27.78%	43.33%	-27.78%	45.95%	-26.49%
Avg		**46.00%**	**-16.36%**	**51.52%**	**-13.26%**	**47.50%**	**-22.88%**

5.4 Conclusion

We presented a multi-class support vector machine based approach to enhance exact multi-objective binary linear programming algorithms. Our approach simulates the best selection of objective function to be used for projection in the KSA in order to improve its computational time. We introduced a pre-ordering approach for the objective functions in the input file for the purpose of standardizing the vector of features. Moreover, we introduced a bi-objective optimization approach for selecting the best subset of features in order to overcome overfitting. By conducting an extensive computational , we showed that reaching to the prediction accuracy of around 70% is possible for instances of tri-objective AP and KP. It was shown that such a prediction accuracy results in a decrease of over 12% in the computational time for some instances.

Overall, we hope that the simplicity of our proposed ML technique and its promising results encourage more researchers to use ML techniques for improving multi-objective optimization solvers.

Chapter 6: Conclusions and Future Research Directions

We now present a summary of the conclusions of this thesis, and some still-open interesting research questions derived from our contributions.

6.1 Optimization Over the Efficient Set

We presented the first criterion space search algorithm for optimizing a linear function over the set of efficient solutions for BOMILP. Our proposed algorithm is built on top of the Triangle Splitting Method (TSM). Therefore, the first research question open for discussion would be, can we build a criterion space search algorithm for optimization over the efficient set on top of an exact algorithm for BOMILP other than the TSM? Additionally, we encourage the creation of decision space algorithms to solve the same problem.

Another open research question is related to the creation of an algorithm for optimization over the efficient set for MOMILP. This includes working with 2 or more objective functions. We believe that such advances will be possible once the first algorithm for finding the nondominated frontier of MOMILP problems becomes available.

6.2 Ecological Applications of the Nash Bargaining Solution

We presented a Nash bargaining solution approach for spatial conservation planning problems modeled using Modern Portfolio Theory (MPT). Our approach is generic and can be used in many other problems of the ecological literature. For instance, one of the interesting real problems, that can be modeled with MPT and solved with our Nash bargaining solution approach, is to determine the appropriate speed limit for boats in the rivers of Florida. The main objective of this problem is to minimize the risk of collision between boats and wild manatees. Another real ecological application is to determine the amount and location of law enforcement troops in some regions of Africa to minimize the risk of poaching. Solving these, and many other related problems, are important for the conservation of biodiversity, and MPT and our Nash bargaining solution approach allow to solve large instances of these problems accurately.

6.3 Learning to Project in MOBLP

Finally, we developed a machine learning approach that learns the best selection of objective function to project in MOBLP. We addressed one of the two open questions about projection in the criterion space. Therefore, the first question that remains open is, can we develop a machine learning approach to learn how to project (what projection method) in MOBLP? Evidently, the question addressed in this thesis and the question that remains open can be extended to MOMILP.

As a reminder, our approach selects an objective function which is fixed for projection during the course of the algorithm. However, criterion space search algorithms usually perform a projection in the reduced objective space at every iteration. Therefore, it is possible to update the learning function at each iteration based on the new information (proportion of the criterion space explored in the previous iteration, and (possible) new nondominated point found) to always select the best objective for projection.

References

[1] 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.

[2] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[3] Mitchell Eaton, Simeon Yurek, Zulqarnain Haider, Julien Martin, Fred Johnson, Bradley Udell, Hadi Charkhgard, and Changhyun Kwon. Spatial conservation planning under uncertainty: adapting to climate change risks using modern portfolio theory. *Ecological Applications*, 2019. . `https://doi.org/10.1002/eap.1962`.

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

[5] Kerstin Dächert and Kathrin Klamroth. A linear bound on the number of scalarizations needed to solve discrete tricriteria optimization problems. *Journal of Global Optimization*, pages 1–34, 2014.

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

[7] 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.

[8] Banu Lokman and Murat Köksalan. Finding all nondominated points of multi-objective integer programs. *Journal of Global Optimization*, 57(2):347–365, 2013.

[9] M Özlen, B A Burton, and C A G MacRae. Multi-objective integer programming: An improved recursive algorithm. *Journal of Optimization Theory and Applications*, 2013. 10.1007/s10957-013-0364-y.

[10] Anthony Przybylski and Xavier Gandibleux. Multi-objective branch and bound. *European Journal of Operational Research*, 260(3):856 – 872, 2017.

[11] Anthony Przybylski, Xavier Gandibleux, and Matthias Ehrgott. A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives. *Discrete Optimization*, 7(3):149 – 165, 2010.

[12] Banu Soylu and Gazi Bilal Yıldız. An exact algorithm for biobjective mixed integer linear programming problems. *Computers & Operations Research*, 72:204–213, 2016.

[13] Natashia Boland, Hadi Charkhgard, and Martin Savelsbergh. A criterion space search algorithm for biobjective integer programming: The balanced box method. *INFORMS Journal on Computing*, 27(4):735–754, 2015.

[14] Natashia Boland, Hadi Charkhgard, and Martin Savelsbergh. The L-shape search method for triobjective integer programming. *Mathematical Programming Computation*, 8(2):217–251, 2016.

[15] Natashia Boland, Hadi Charkhgard, and Martin Savelsbergh. The quadrant shrinking method: A simple and efficient algorithm for solving tri-objective integer programs. *European Journal of Operational Research*, 2016. available online.

[16] 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.

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

[18] 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*, 2016. available online.

[19] Murat Köksalan and Banu Lokman. Finding nadir points in multi-objective integer programs. *Journal of Global Optimization*, 62(1):55–77, May 2015.

[20] Gokhan Kirlik and Serpil Sayın. Computing the nadir point for multiobjective discrete optimization problems. *Journal of Global Optimization*, 62(1):79–99, May 2015.

[21] Özgür Özpeynirci. On nadir points of multiobjective integer programming problems. *Journal of Global Optimization*, May 2017. available online.

[22] 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.

[23] 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.

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

[25] Harold P Benson. An all-linear programming relaxation algorithm for optimizing over the efficient set. *Journal of Global Optimization*, 1(1):83–104, 1991.

[26] Harold P Benson. A finite, nonadjacent extreme-point search algorithm for optimization over the efficient set. *Journal of Optimization Theory and Applications*, 73(1):47–64, 1992.

[27] Harold P Benson. A bisection-extreme point search algorithm for optimizing over the efficient set in the linear dependence case. *Journal of Global Optimization*, 3(1):95–111, 1993.

[28] Jerald P Dauer. Optimization over the efficient set using an active constraint approach. *Mathematical Methods of Operations Research*, 35(3):185–195, 1991.

[29] JG Ecker and JH Song. Optimizing a linear function over an efficient set. *Journal of Optimization Theory and Applications*, 83(3):541–563, 1994.

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

[31] Yoshitsugu Yamamoto. Optimization over the efficient set: overview. *Journal of Global Optimization*, 22(1-4):285, 2002.

[32] Moncef Abbas and Djamal Chaabane. Optimizing a linear function over an integer efficient set. *European Journal of Operational Research*, 174(2):1140–1161, 2006.

[33] Chaabane Djamal and Pirlot Marc. A method for optimizing over the integer efficient set. *Journal of industrial and management optimization*, 6(4):811, 2010.

[34] Djamal Chaabane, B Brahmi, and Z Ramdani. The augmented weighted tchebychev norm for optimizing a linear function over an integer efficient set of a multicriteria linear program. *International Transactions in Operational Research*, 19(4):531–545, 2012.

[35] Ali Fattahi and Metin Turkay. A one direction search method to find the exact nondominated frontier of biobjective mixed-binary linear programming problems. *European Journal of Operational Research*, 2017. avaiable online.

[36] T. Vincent, F Seipp, S Ruzika, A Przybylski, and X Gandibleux. Multiple objective branch and bound for mixed 0-1 linear programming: Corrections and improvements for biobjective case. *Computers & Operations Research*, 40(1):498–509, 2013.

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

[38] P Belotti, B Soylu, and MM Wiecek. A branch-and-bound algorithm for biobjective mixed-integer programs. 2013. `http://www.optimization-online.org/DB_FILE/2013/01/3719.pdf`. Last accessed: September 23, 2017.

[39] Iain Dunning, Joey Huchette, and Miles Lubin. Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[40] Aritra Pal and Hadi Charkhgard. FPBH. jl: A feasibility pump based heuristic for multi-objective mixed integer linear programming in julia. 2018. `http://www.optimization-online.org/DB_FILE/2017/09/6195.pdf`. Last accessed: March 01, 2019.

[41] Wei-Jie Yu, Jin-Zhou Li, Wei-Neng Chen, and Jun Zhang. A parallel double-level multiobjective evolutionary algorithm for robust optimization. *Applied Soft Computing*, 59:258 – 275, 2017.

[42] William Pettersson and Melih Özlen. A parallel approach to bi-objective integer programming. *ANZIAM Journal*, 58:69–81, 2017.

[43] Xavier Gandibleux, Gauthier Soleilhac, Anthony Przybylski, Flavien Lucas, Stefan Ruzika, and Pascal Halffmann. voptsolver, a "get and run" solver of multiobjective linear optimization problems built on julia and jump. *MCDM2017: 24th International Conference on Multiple Criteria Decision Making*, 2017.

[44] Aritra Pal and Hadi Charkhgard. MSEA. jl: A multi-stage exact algorithm for bi-objective pure integer linear programming in julia. 2018. `http://www.optimization-online.org/DB_FILE/2018/04/6595.pdf`. Last accessed: March 01, 2019.

[45] Yicheng Wang and Hayri Önal. Optimal design of compact and connected nature reserves for multiple species. *Conservation biology*, 30(2):413–424, 2016.

[46] RL Pressey, CJ Humphries, Christopher R Margules, RI Vane-Wright, and PH Williams. Beyond opportunism: key principles for systematic reserve selection. *Trends in ecology & evolution*, 8(4):124–128, 1993.

[47] April E Reside, Nathalie Butt, and Vanessa M Adams. Adapting systematic conservation planning for climate change. *Biodiversity and Conservation*, 27(1):1–29, 2018.

[48] Sharon E Kingsland. Creating a science of nature reserve design: perspectives from history. *Environmental Modeling & Assessment*, 7(2):61–69, 2002.

[49] James Boyd, Rebecca Epanchin-Niell, and Juha Siikamäki. Conservation planning: a review of return on investment analysis. *Review of Environmental Economics and Policy*, 9(1):23–42, 2015.

[50] Mark W Schwartz, Carly N Cook, Robert L Pressey, Andrew S Pullin, Michael C Runge, Nick Salafsky, William J Sutherland, and Matthew A Williamson. Decision support frameworks and tools for conservation. *Conservation Letters*, 11(2):e12385, 2018.

[51] 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.

[52] Jessica Zamborain-Mason, Garry R Russ, Rene A Abesamis, Abner A Bucol, and Sean R Connolly. Network theory and metapopulation persistence: incorporating node self-connections. *Ecology letters*, 20(7):815–831, 2017.

[53] Benjamin S Ramage, Elaina C Marshalek, Justin Kitzes, and Matthew D Potts. Conserving tropical biodiversity via strategic spatiotemporal harvest planning. *Journal of Applied Ecology*, 50(6):1301–1310, 2013.

[54] Christopher Costello and Stephen Polasky. Dynamic reserve site selection. *Resource and Energy Economics*, 26(2):157–174, 2004.

[55] Adam W Schapaugh and Andrew J Tyre. A simple method for dealing with large state spaces. *Methods in Ecology and Evolution*, 3(6):949–957, 2012.

[56] Hayri Önal and Yicheng Wang. A graph theory approach for designing conservation reserve networks with minimal fragmentation. *Networks: An International Journal*, 51(2):142–152, 2008.

[57] Ian R Ball, Hugh P Possingham, and M Watts. Marxan and relatives: software for spatial conservation prioritisation. *Spatial conservation prioritisation: Quantitative methods and computational tools*, pages 185–195, 2009.

[58] Noam Levin, James EM Watson, Liana N Joseph, Hedley S Grantham, Liat Hadar, Naomi Apel, Avi Perevolotsky, Niv DeMalach, Hugh P Possingham, and Salit Kark. A framework for systematic conservation planning and management of mediterranean landscapes. *Biological Conservation*, 158:371–383, 2013.

[59] Yicheng Wang, Hayri Önal, and Qiaoling Fang. How large spatially-explicit optimal reserve design models can we solve now? an exploration of current models' computational efficiency. *Nature Conservation*, 27:17, 2018.

[60] Justin C Williams, Charles S ReVelle, and Simon A Levin. Using mathematical optimization models to design nature reserves. *Frontiers in Ecology and the Environment*, 2(2):98–105, 2004.

[61] Rebecca K Runting, Catherine E Lovelock, Hawthorne L Beyer, and Jonathan R Rhodes. Costs and opportunities for preserving coastal wetlands under sea level rise. *Conservation Letters*, 10(1):49–57, 2017.

[62] Diogo Alagador, Jorge Orestes Cerdeira, and Miguel Bastos Araújo. Climate change, species range shifts and dispersal corridors: an evaluation of spatial conservation models. *Methods in Ecology and Evolution*, 7(7):853–866, 2016.

[63] Neville D Crossman and Brett A Bryan. Systematic landscape restoration using integer programming. *Biological Conservation*, 128(3):369–383, 2006.

[64] Olga Chernomor, Bui Quang Minh, Félix Forest, Steffen Klaere, Travis Ingram, Monika Henzinger, and Arndt von Haeseler. Split diversity in constrained conservation prioritization using integer linear programming. *Methods in ecology and evolution*, 6(1):83–91, 2015.

[65] Rodolfo Carvajal, Miguel Constantino, Marcos Goycoolea, Juan Pablo Vielma, and Andrés Weintraub. Imposing connectivity constraints in forest planning models. *Operations Research*, 61(4):824–836, 2013.

[66] Paul R Armsworth, Szvetlana Acs, Martin Dallimer, Kevin J Gaston, Nick Hanley, and Paul Wilson. The cost of policy simplification in conservation incentive programs. *Ecology letters*, 15(5):406–414, 2012.

[67] John C Withey, Joshua J Lawler, Stephen Polasky, Andrew J Plantinga, Erik J Nelson, Peter Kareiva, Chad B Wilsey, Carrie A Schloss, Theresa M Nogeire, Aaron Ruesch, et al. Maximising return on conservation investment in the conterminous usa. *Ecology Letters*, 15(11):1249–1256, 2012.

[68] Sahan TM Dissanayake, Hayri Önal, James D Westervelt, and Harold E Balbach. Incorporating species relocation in reserve design models: An example from ft. benning ga. *Ecological modelling*, 224(1):65–75, 2012.

[69] Sahan TM Dissanayake and Hayri Önal. Amenity driven price effects and conservation reserve site selection: A dynamic linear integer programming approach. *Ecological Economics*, 70(12):2225–2235, 2011.

[70] Bronwyn Rayfield, David Pelletier, Maria Dumitru, Jeffrey A Cardille, and Andrew Gonzalez. Multipurpose habitat networks for short-range and long-range connectivity: a new method combining graph and circuit connectivity. *Methods in Ecology and Evolution*, 7(2):222–231, 2016.

[71] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[72] Matthew E Watts, Ian R Ball, Romola S Stewart, Carissa J Klein, Kerrie Wilson, Charles Steinback, Reinaldo Lourival, Lindsay Kircher, and Hugh P Possingham. Marxan with zones: software for optimal conservation based land-and sea-use zoning. *Environmental Modelling & Software*, 24(12):1513–1521, 2009.

[73] Ayesha IT Tulloch, Richard F Maloney, Liana N Joseph, Joseph R Bennett, Martina MI Di Fonzo, William JM Probert, Shaun M O'connor, Jodie P Densem, and Hugh P Possingham. Effect of risk aversion on prioritizing conservation projects. *Conservation Biology*, 29(2):513–524, 2015.

[74] Bradley J Udell, Julien Martin, Robert J Fletcher Jr, Mathieu Bonneau, Holly H Edwards, Timothy A Gowan, Stacie K Hardy, Eliezer Gurarie, Charles S Calleson, and Charles J Deutsch. Integrating encounter theory with decision analysis to evaluate collision risk and determine optimal protection zones for wildlife. *Journal of Applied Ecology*, 2018.

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

[76] Benjamin S Halpern, Crow White, Sarah E Lester, Christopher Costello, and Steven D Gaines. Using portfolio theory to assess tradeoffs between return from natural capital and social equity across space. *Biological Conservation*, 144(5):1499–1507, 2011.

[77] Mindy L Mallory and Amy W Ando. Implementing efficient conservation portfolio design. *Resource and Energy Economics*, 38:1–18, 2014.

[78] Sergio Alvarez, Sherry L Larkin, and Andrew Ropicki. Optimizing provision of ecosystem services using modern portfolio theory. *Ecosystem services*, 27:25–37, 2017.

[79] John F Nash Jr. The bargaining problem. *Econometrica: Journal of the Econometric Society*, pages 155–162, 1950.

[80] 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.

[81] Harry Markowitz. Portfolio selection. *The journal of finance*, 7(1):77–91, 1952.

[82] Harry M. Markowitz. *Portfolio Selection: Efficient Diversification of Investment*. John Wiley & Sons, New York, 1959.

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

[84] Elias Boutros Khalil, Pierre Le Bodic, Le Song, George L Nemhauser, and Bistra N Dilkina. Learning to branch in mixed integer programming. In *AAAI*, pages 724–731, 2016.

[85] Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195, 2017.

[86] Ashish Sabharwal, Horst Samulowitz, and Chandra Reddy. Guiding combinatorial optimization with uct. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 356–361. Springer, 2012.

[87] He He, Hal Daume III, and Jason M Eisner. Learning to search in branch and bound algorithms. In *Advances in neural information processing systems*, pages 3293–3301, 2014.

[88] Elias B Khalil, Bistra Dilkina, George L Nemhauser, Shabbir Ahmed, and Yufen Shao. Learning to run heuristics in tree search. In *Proceedings of the international joint conference on artificial intelligence. AAAI Press, Melbourne, Australia*, 2017.

[89] Dan Roth and Wen-tau Yih. Integer linear programming inference for conditional random fields. In *Proceedings of the 22nd international conference on Machine learning*, pages 736–743. ACM, 2005.

[90] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[91] Quoc V Le, Jiquan Ngiam, Adam Coates, Abhik Lahiri, Bobby Prochnow, and Andrew Y Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 265–272. Omnipress, 2011.

[92] Suvrit Sra, Sebastian Nowozin, and Stephen J Wright. *Optimization for machine learning*. MIT Press, 2012.

[93] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[94] Dimitris Bertsimas, Angela King, Rahul Mazumder, et al. Best subset selection via a modern optimization lens. *The annals of statistics*, 44(2):813–852, 2016.

[95] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of machine learning research*, 2(Dec):265–292, 2001.

[96] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, page 104. ACM, 2004.

[97] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[98] Anita Prinzie and Dirk Van den Poel. Random forests for multiclass classification: Random multinomial logit. *Expert systems with Applications*, 34(3):1721–1732, 2008.

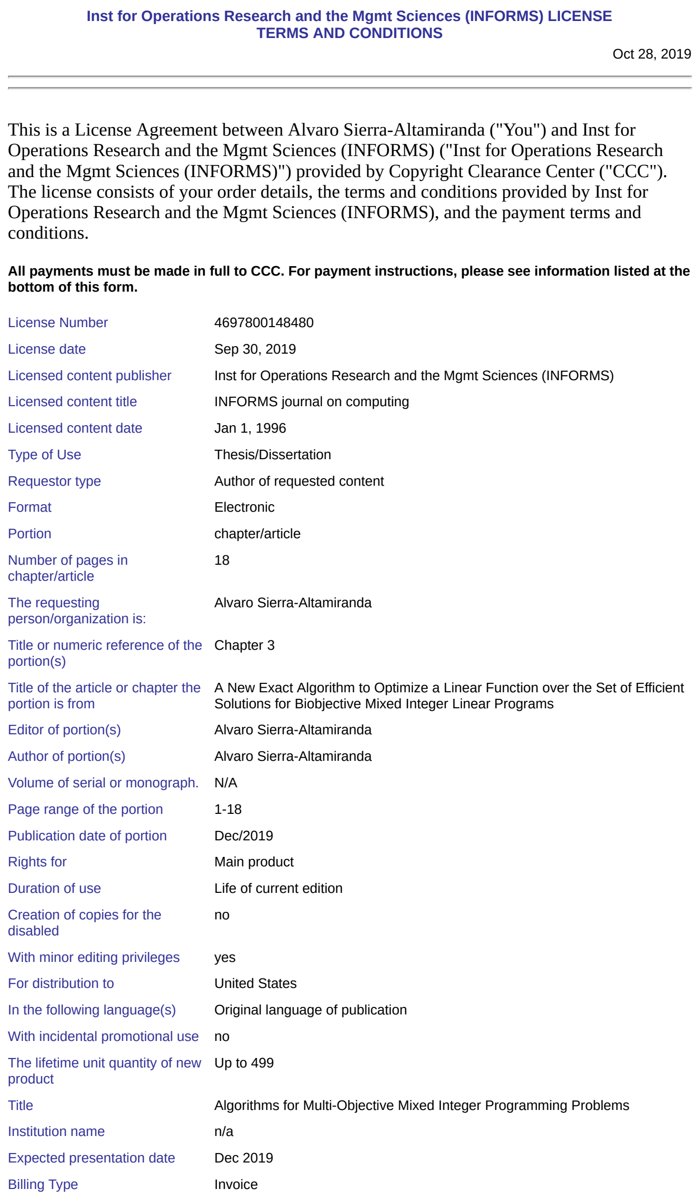
[99] Hadi Charkhgard and Ali Eshragh. A new approach to select the best subset of predictors in linear regression modeling: bi-objective mixed integer linear programming. *ANZIAM journal*, 2019. . Available online. `https://doi.org/10.1017/S1446181118000275`.

[100] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1996.

[101] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Appendix A: Copyright Permissions

Appendix A1: Reprint Permission for Chapter 3, Sections 3.1 - 3.3



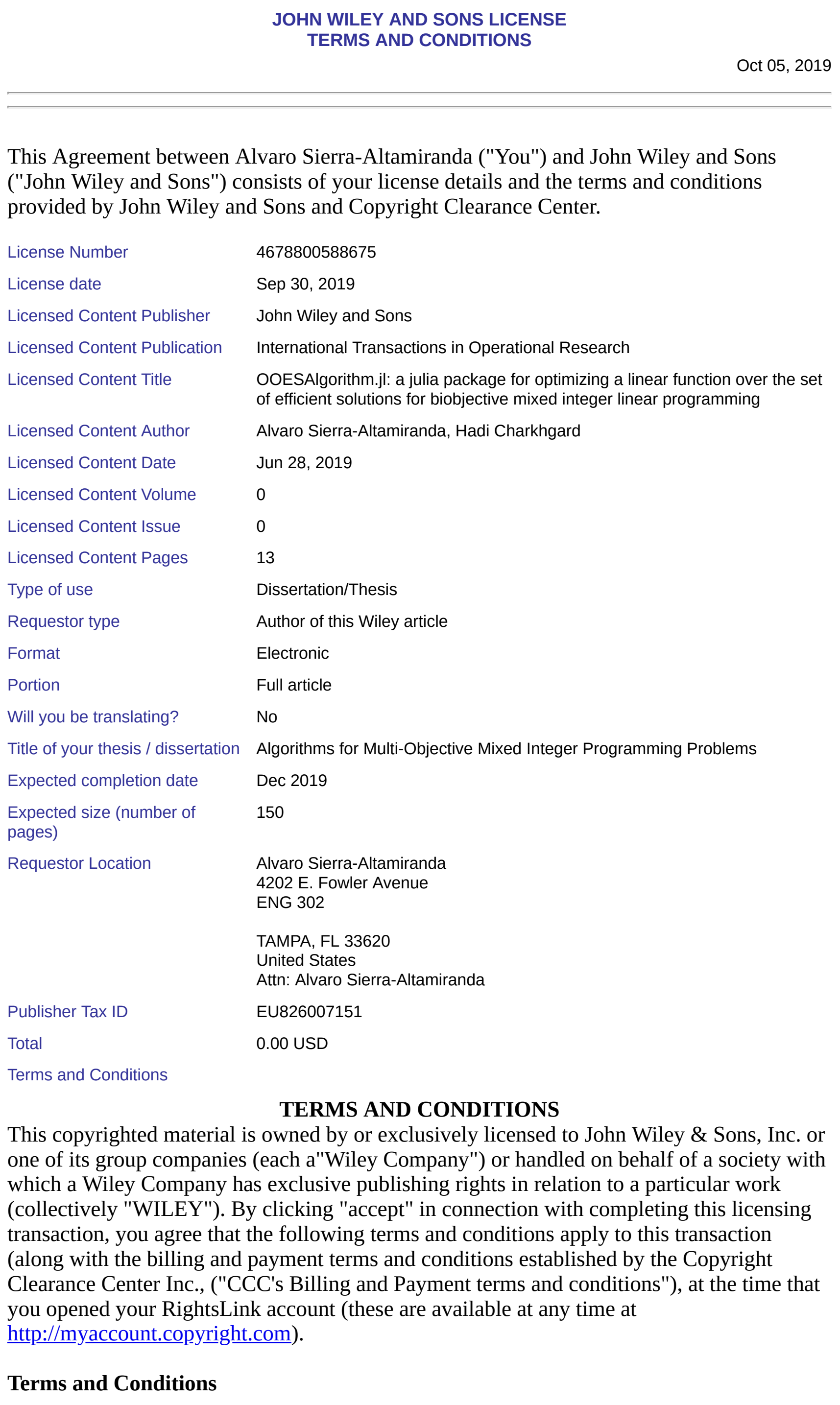
Inst for Operations Research and the Mgmt Sciences (INFORMS) LICENSE TERMS AND CONDITIONS

Oct 28, 2019

License Number	4697800148480
License date	Sep 30, 2019
Licensed content publisher	Inst for Operations Research and the Mgmt Sciences (INFORMS)
Licensed content title	INFORMS journal on computing
Licensed content date	Jan 1, 1996
Type of Use	Thesis/Dissertation
Requestor type	Author of requested content
Format	Electronic
Portion	chapter/article
Number of pages in chapter/article	18
The requesting person/organization is:	Alvaro Sierra-Altamiranda
Title or numeric reference of the portion(s)	Chapter 3
Title of the article or chapter the portion is from	A New Exact Algorithm to Optimize a Linear Function over the Set of Efficient Solutions for Biobjective Mixed Integer Linear Programs
Editor of portion(s)	Alvaro Sierra-Altamiranda
Author of portion(s)	Alvaro Sierra-Altamiranda
Volume of serial or monograph.	N/A
Page range of the portion	1-18
Publication date of portion	Dec/2019
Rights for	Main product
Duration of use	Life of current edition
Creation of copies for the disabled	no
With minor editing privileges	yes
For distribution to	United States
In the following language(s)	Original language of publication
With incidental promotional use	no
The lifetime unit quantity of new product	Up to 499
Title	Algorithms for Multi-Objective Mixed Integer Programming Problems
Institution name	n/a
Expected presentation date	Dec 2019
Billing Type	Invoice

Billing Address	Alvaro Sierra-Altamiranda 4202 E. Fowler Avenue ENG 302 TAMPA, FL 33620 United States Attn: Alvaro Sierra-Altamiranda
Total (may include CCC user fee)	0.00 USD
Terms and Conditions	

Appendix A2: Reprint Permission for Chapter 3, Section 3.4



JOHN WILEY AND SONS LICENSE
TERMS AND CONDITIONS

Oct 05, 2019

This Agreement between Alvaro Sierra-Altamiranda ("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	4678800588675
License date	Sep 30, 2019
Licensed Content Publisher	John Wiley and Sons
Licensed Content Publication	International Transactions in Operational Research
Licensed Content Title	OOESAlgorithm.jl: a julia package for optimizing a linear function over the set of efficient solutions for biobjective mixed integer linear programming
Licensed Content Author	Alvaro Sierra-Altamiranda, Hadi Charkhgard
Licensed Content Date	Jun 28, 2019
Licensed Content Volume	0
Licensed Content Issue	0
Licensed Content Pages	13
Type of use	Dissertation/Thesis
Requestor type	Author of this Wiley article
Format	Electronic
Portion	Full article
Will you be translating?	No
Title of your thesis / dissertation	Algorithms for Multi-Objective Mixed Integer Programming Problems
Expected completion date	Dec 2019
Expected size (number of pages)	150
Requestor Location	Alvaro Sierra-Altamiranda 4202 E. Fowler Avenue ENG 302 TAMPA, FL 33620 United States Attn: Alvaro Sierra-Altamiranda
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

Appendix B: Chapter 3

Appendix B1: Documentation of `OOESAlgorithm.jl`

Characteristics: Some important characteristics of this package are:

- The package can solve any (both structured and unstructured) biobjective mixed integer linear problem. The following problem classes are supported:
  1. Objectives: 2 linear objectives and one extra objective to be optimized over the efficient set.
  2. Constraints: 0 or more linear (both inequality and equality) constraints
  3. Variables:
     - Binary variables
     - Integer variables
     - Continuous variables
- A biobjective mixed integer linear instance can be provided as input to the package in 3 ways:
  1. `JuMP` Model
  2. LP file format
  3. MPS file format
- Any mixed integer linear programming solver supported by MathProgBase.jl can be used.

- `OOESAlgorithm.jl` automatically installs `GLPK.jl` as the default single-objective mixed integer linear programming solver. If the user desires to use any other MIP solver, it must be separately installed. Note that `OOESAlgorithm.jl` has been successfully tested with:
  1. GLPK - v4.61
  2. SCIP - v5.0.1
     * The file /.julia/packages/SCIP/.../src/mpb_interface.jl of the package SCIP.jl v0.6.1 has to be modified manually. In lines 185, 186, 192, 193, 207 and 208, numvar(m) has to be changed by MathProgBase.numvar(m).
  3. Gurobi - v7.5
  4. CPLEX - v12.7

- Supports parallelization.
  - Can use how many threads are available when julia is opened.
  - Can implement different parallelization types.
- Can be easily modified to compute the entire Pareto-optimal frontier.
  - Can compute the nondominated frontier by just tuning a parameter.

Installation: Next you will find the instruuctions to install `OOESAlgorithm.jl`.

`OOESAlgorithm.jl` is a package for Julia. To use `OOESAlgorithm.jl`, first download and install Julia. Note that the whole ecosystem has been tested using Julia v1.0.2 and hence we can cannot guarantee whether it will work with previous versions of Julia. Thus, it is important that a Julia version v1.0.2 or higher is properly installed and available on your system. Our package is available at the `METADATA.jl` repository, therefore, from Julia, the latest released version of `OOESAlgorithm.jl` is installed by using the package environment:

```
(v1.0) pkg> add OOESAlgorithm
```

You may also install the package with the latest updates using the built-in package manager:

```
julia> import Pkg
julia> Pkg.clone(“https://github.com/alvsierra286/OOESAlgorithm.jl")
julia> Pkg.build(“OOESAlgorithm")
```

The dependencies of `OOESAlgorithm.jl` include the following Julia packages:

- `MathOptInterface.jl`
- `MathProgBase.jl`
- `JuMP.jl`
- `GLPKMathProgInterface.jl`
- `GLPK.jl`

Note that `GLPK.jl` is installed with the package, and therefore, used as the default single-objective mixed integer programming (MIP) solver. However, our package relies on the MIP solvers supported by the package `MathProgBase.jl`. In this sense (if available) we strongly recommend to use other optimization solvers.

Next you will find a list of MIP solvers supported by `OOESAlgorithm.jl`.

Observe that some of the supported solvers are available for public and academic use. We refer the users to the links in the table of supported solvers to follow guidelines for correct installation.

Getting Started: Start guide to introduce the main concepts of `OOESAlgorithm.jl`.

Once `OOESAlgorithm.jl` is installed, to use `OOESAlgorithm.jl` in your programs, you just need to say:

```
julia> using OOESAlgorithm
```

Solver	**Julia Package**	**License**
Cbc	`Cbc.jl`	Eclipse Public License
CPLEX	`CPLEX.jl`	Commercial
FICO Xpress	`Xpress.jl`	Commercial
GLPK	`GLPK.jl`	General Public License
Gurobi	`Gurobi.jl`	Commercial
MOSEK	`MathProgBaseMosek.jl`	Commercial
SCIP	`SCIP.jl`	ZIB Academic License

The main function exported when using `OOESAlgorithm.jl` is called OOES, and returns the optimal solution for the problem of optimizing a linear function over the set of efficient solutions for biobjective mixed integer linear programs. To use this function, we need to provide the input in file format (LP, MPS) or a `JuMP` model. For example:

- LP format:

```
julia> OOES(“input_file.lp")
```

- MPS format:

```
julia> OOES(“input_file.mps")
```

- `JuMP` model:

```
julia> model = JuMP.Model()
after creating the model:
julia> OOES(model)
```

`OOESAlgorithm.jl` assumes that all the functions are in minimization form. In further section we explain how to consider problems in maximization form.

`OOESAlgorithm.jl` automatically creates an output summary file with the value of the objective functions and a solution file with the value of the variables. We can also assign the optimal solution to a variable:

```
julia> optimal_solution = OOES(model)
```

The solution returned has the following information:

```
vector with the values of the objective functions:
julia> optimal_solution.obj_vals

vector with the values of the variables:
julia> optimal_solution.vars

true when solution is optimal over a nondominated point, false otherwise:
julia> optimal_solution.fxopt
```

Consider the following BOMILP problem:

$$\begin{aligned} &\min\ 3x_1 + x_2 \\ &\min\ -x_1 - 2x_2 \\ \text{subject to: } &x_2 \leq 3 \\ &3x_1 - x_2 \leq 6 \\ &x1 \in \mathbb{R}_+,\ x_2 \in \mathbb{Z}_+, \end{aligned}$$

and let the linear function we want to optimize over the set of efficient solutions of the problem be:

$$-2x_1 + 3x_2.$$

To represent this problem in any format, we first consider a generic minimization function equivalent to the sum of all the variables. The order of the variables in this objective function will correspond to the order of the variables in the output. Then we add the constraints of the problem, and finally we add the 2 objective functions and the function to be optimized over the efficient set as additional equality constraints with RHS equal to 0. The first two functions will be the objective functions of the BOMILP program and the third function will be the linear function to be optimized over the efficient set. In some cases, when using a `JuMP` model, objective functions can be confused with some other equality constraints with RHS equal to 0. In this case, replace the equality constraint with two constraints (one $\geq$ and the other $\leq$).

The following box represents the text of the BOMILP problem in LP file format:

```
Minimize
obj: x1+x2
Subject To
c1: x2 <= 3
c2: 3 x1 - x2 <= 6
c3: 3 x1 + x2 = 0
c4: - x1 - 2 x2 = 0
c5: -2 x1 + 3 x2 = 0

Bounds
0 <= x1 <= 1e10
0 <= x2 <= 1e10

General
x2

End
```

The following box represents the text of the BOMILP problem in MPS file format:

```
ROWS
N obj
L c1
L c2
E c3
E c4
E c5
COLUMNS
x1    obj    1
x1    c2     3
x1    c3     3
x1    c4    -1
x1    c5    -2
MARK0000    'MARKER'    'INTORG'
x2          obj          1
x2          c1           1
x2          c2          -1
x2          c3           1
x2          c4          -2
x2          c5           3
MARK0001    'MARKER'    'INTEND'
RHS
rhs         c1           3
rhs         c2           6
BOUNDS
UP bnd x1 10000000000
UP bnd x2 10000000000
ENDATA
```

And finally, the following box shows the julia commands to represent the BOMILP as a `JuMP` model:

```
julia> using JuMP
julia> model = JuMP.Model()
julia> @variable(model, 0 <= x1 <= 1e10)
julia> @variable(model, 0 <= x2 <= 1e10, Int)
julia> @objective(model, Min, x1 + x2)
julia> @constraint(model, x2 <= 3)
julia> @constraint(model, 3x1 - x2 <= 6)
julia> @constraint(model, 3x1 + x2 == 0)
julia> @constraint(model, -x1 - 2x2 == 0)
julia> @constraint(model, -2x1 + 3x2 == 0)
```

Advanced Features: In this section we explain some of the advanced features of `OOESAlgorithm.jl`. The advanced features are divided in tuning parameters, solver selection and parallelization. Additionally, we include some test files that run our package under some scenarios with the default parameters. For instance, lets assume that our input is a JuMP.Model() called model.

Time Limit: Time limit is a termination condition of `OOESAlgorithm.jl`. When the running time of `OOESAlgorithm.jl` reach the time limit, it automatically stops. By default, `OOESAlgorithm.jl` finish the current iteration before exiting. The default value for the time limit is 86400.0 seconds, which is equivalent to one day. The following command must be followed in order to modify the time limit:

```
julia> OOES(model, timelimit=α)
```

where α is a floating number with the new time limit in seconds.

For example, if we want to run our package for one hour, then we must type:

```
julia> OOES(model, timelimit=3600.0)
```

Relative Gap: The relative gap is a termination condition of the single-objective MIP solver implemented in `OOESAlgorithm.jl`. The relative gap by default is 1.0^{-6}. To change this value, we must type:

```
julia> OOES(model, relative_gap=β)
```

where β is a floating number with the new relative gap.

For example, if we want to run our package with a relative gap of 1.0^{-9}, then we must type:

```
julia> OOES(model, relative_gap=1.0e-9)
```

Optimization Sense: The optimization sense is a parameter that let us declare if the objective function is in minimization or maximization form. By default, the sense of all objective functions is minimization, however, we can change that by tuning the vector sense in `OOESAlgorithm.jl`. For example, if our first objective function is in minimization form, while the second and third functions are in maximization form, we must type:

```
julia> OOES(model, sense=[:Min, :Max, :Max])
```

Solver Selection: `OOESAlgorithm.jl` offers two different options to select the single-objective MIP solver to implement. One of the options is by tuning the parameter mipsolver, and the other by tuning the parameter mip_solver.

mipsolver: `OOESAlgorithm.jl` automatically detects if some of the most popular solvers are installed, and mipsolver represents one of the solvers with an integer number. The list of the solvers with their respective mipsolver number is shown next:

For example, if we want to run our package with `CPLEX.jl` (if CPLEX is available and properly installed), then we must type:

```
julia> OOES(model, mipsolver=3)
```

Solver	mipsolver
`GLPK.jl`	1
`Gurobi.jl`	2
`CPLEX.jl`	3
`SCIP.jl`	4
`Xpress.jl`	5

mip_solver: We can tune the parameter mip_solver when using `OOESAlgorithm.jl` to explore advanced settings in the single-objective MIP solvers. This feature is recommended for advanced users. A list with some of the solvers with their respective mip_solver code is shown next:

Solver	mip_solver
`Gurobi.jl`	GurobiSolver(OutputFlag=ϕ, Threads=ρ, MIPGap=γ)
`CPLEX.jl`	CplexSolver(CPX_PARAM_SCRIND=ϕ, CPX_PARAM_THREADS=ρ, CPX_PARAM_EPGAP=γ)
`SCIP.jl`	SCIPSolver("display/verblevel", ϕ, "limits/gap", γ)
`Xpress.jl`	Xpress.XpressSolver(XPRS_OUTPUTLOG=ϕ, XPRS_THREADS=ρ, XPRS_STOP_MIPGAP=γ)

where ϕ indicates the level of verbosity of the MIP solver, ρ is the number of threads (if multiple processors are available, we strongly recommend to declare $\rho = 1$ and exploit parallelization by tuning the parallelization parameters), and γ is the relative gap. For example, if we want to run our package with `Gurobi.jl` (Given that Gurobi is available and properly installed), with no screen information, 1 thread and relative gap of 1.0^{-12} then we must type:

```
julia> OOES(model, mip_solver=GurobiSolver(OutputFlag=0, Threads=1, MIPGap=1e-12))
```

Note that, if the solver is not detected by the package, it must be added manually as shown in the next example:

```
julia> using OOESAlgorithm, MathProgBaseMosek
```

Parallelization: In this section, we explain two important parameters to exploit parallelization in our package. In order to enable multiple processors, we must type in the terminal the following code:

```
~ $ julia -p Ω
```

where Ω is the number of processors to enable.

For example, if we want to enable 3 processors, then we must type:

```
~ $ julia -p 3
```

Number Of Threads: Once we enable multiple processors, we can tune a parameter to declare the number of threads we want to use. This parameter should have a value lesser or equal to the number of available processors. The following command must be followed in order to modify the number of threads:

```
julia> OOES(model, threads=σ)
```

where σ is an integer number with the new number of threads to use.

For example, if we want to run our package with three threads, then we must type:

```
julia> OOES(model, threads=3)
```

Parallelization Techniques: We can tune a parameter to exploit different parallelization techniques. There are a total of 4 techniques, three based on cuts in the criterion space and one based on the elements of the priority queue of the algorithm. To choose between the different parallelization techniques, we tune the parameter parallelization, which is an integer number. The list parallelization techniques with their respective parameter value are shown next:

The following command must be followed in order to choose a different parallelization technique:

Technique	**parallelization**
Horizontal cuts	1
Vertical cuts	2
Diagonal cuts	3
Elements of the Priority Queue	4

```
julia> OOES(model, parallelization=θ)
```

where θ is an integer number that represents the parallelization technique to use.

For example, if we want to run our package, using diagonal cuts, then we must type:

```
julia> OOES(model, parallelization=3)
```

Computing The Pareto-Optimal Frontier: The package can easily be modified to compute the entire Pareto-optimal frontier of a biobjective mixed integer linear programming. The name of the parameter to tune is pareto_frontier, a boolean variable that indicates that the package will compute the Pareto-optimal frontier when the value is true. For example, if we want to get the Pareto-optimal frontier, using our package, we must type:

```
julia> OOES(model, pareto_frontier=true)
```

The algorithm implemented to compute the Pareto-optimal frontier is the *Triangle Splitting Method*. The output of the package contains the points in the Pareto-optimal frontier, organized in non-decreasing order of the first objective function, and a boolean parameter that indicates if such point is connected to the next point in the frontier by a line or not. If connected, then the connection is a line contained in the Pareto-optimal frontier.

Please be aware of the following observation: to be sure that our package is computing the exact nondominated frontier, the input file must have 3 objectives just as if we were optimizing the linear function over the efficient set, i.e., 3 constraints equal to 0. The order of the objectives is the same as the explained in the subsection 'Creating The Input For `OOESAlgorithm.jl`'. The third objective function will automatically be obviated by the package, therefore we may consider any simple function. Additionally, the package returns a vector with the points Pareto-optimal frontier and the solutions correspondent to such points. An example to obtain the vector is shown next:

```
julia> Pareto_optimal_frontier = OOES(model, pareto_frontier=true)
```

In this example, the variable Pareto_optimal_frontier is a vector with as many positions as the points discovered in the nondominated frontier. Each position will contain the following information:

vector with the values of the objective functions in the point ξ of the Pareto-optimal frontier:

```
julia> Pareto_optimal_frontier[ξ].obj_vals
```

vector with the values of the variables in the point ξ of the Pareto-optimal frontier:

```
julia> Pareto_optimal_frontier[ξ].vars
```

true when the point ξ is connected to the next point of the Pareto-optimal frontier by a line:

```
julia> Pareto_optimal_frontier[ξ].fxopt
```

The parameter pareto_frontier also supports parallelization, however, it doesn't support the fourth technique, i.e., parallelization=4.

Testing: We provide our package with a test located in OOESAlgorithm.jl/test/ to check the installation. After installing `OOESAlgorithm.jl`, open julia in a new terminal, go to pkg environment and execute the following code:

```
(v1.0) pkg> test OOESAlgorithm
```

The test executes a total of three examples, one of them a JuMP model, and the other two LP and MPS files. If the installation of `OOESAlgorithm.jl` was successful and the solver returns the optimal values for the objective functions, the following message should appear:

```
Solution test 1: [-276.667,  -257.333,  -443.0]
Test 1 Successful


Solution test 2: [-276.667,  -257.333,  -443.0]
Test 2 Successful


Solution test 3: [-276.667,  -257.333,  -443.0]
Test 3 Successful


        Testing OOESAlgorithm tests passed
```

Contributions: This package is written and maintained by Alvaro Sierra-Altamiranda. Please fork and send a pull request or create a GitHub issue for bug reports or feature requests.

Appendix C: Chapter 4

Appendix C1: Neighboring Constraints

B1 For each parcel j, let x_j be a binary decision variable indicating whether a parcel should be selected ($x_j = 1$) or not ($x_j = 0$). Based on this definition, a common constraint when formulating SCP problems is that a given parcel i can be selected only if parcel j is selected. To enforce this, one can implement the following constraint,

$$x_i - x_j \leq 0. \tag{1}$$

In contrast, if the desired result is to protect at most one of the parcels i and j, then the following constraint must be added,

$$x_i + x_j \leq 1. \tag{2}$$

Let C be the set of parcels neighboring a given parcel j. Then, one can define the following constraint to ensure that parcel j can be selected only if at least one of its neighbors is also selected,

$$\sum_{i \in C} x_i - x_j \geq 0. \tag{3}$$

One can also define the following constraint to ensure that parcel j can be selected only if all of its neighbors are also selected,

$$\sum_{i \in C} x_i - |C| x_j \geq 0. \tag{4}$$

Now let $n < |C|$ be an integer number. Then, one can define the following constraint to ensure that parcel j can be selected only if at least n neighbors are also selected,

$$\sum_{i \in C} x_i - n x_j \geq 0. \tag{5}$$

One can also define the following constraint to ensure that parcel j can be selected only if at most n neighbors are also selected,

$$\sum_{i \in C} x_i + (|C| - n) x_j \leq |C|. \tag{6}$$

Finally, one can define the following constraint to ensure that parcel j can be selected only if none of its neighbors is selected,

$$\sum_{i \in C} x_i + |C| x_j \leq |C|. \tag{7}$$

Appendix C2: Data for the Example

In this appendix, we provide the data used for the numerical example in subsection 4.3.2.

Table B2.1 – Parcels data

Parcel	Initial State	Return ($)	Variance ($\times 10^5$)	Size
1	Unprotected	0.1	0.007	453.63
2	Unprotected	0.1	0.009	8.71
3	Unprotected	1.3	950.404	5.89
4	Unprotected	0.1	0.069	278.53
5	Unprotected	0.0	0.012	9.18
6	Protected	1.7	50.037	154.40
7	Unprotected	0.2	0.222	8.54
8	Protected	1.2	115.276	160.80
9	Unprotected	0.1	0.109	9.70
10	Protected	2.8	2,074.640	58.99
11	Unprotected	0.4	0.587	14.08
12	Protected	0.5	98.375	359.54
13	Unprotected	0.1	2,379.336	6.80
14	Protected	0.1	0.000	16.84
15	Unprotected	0.8	20.500	15.94
16	Unprotected	0.5	42.332	15.05
17	Unprotected	0.1	0.051	7.54
18	Unprotected	0.1	0.420	109.45
19	Unprotected	0.0	0.004	89.77
20	Protected	0.4	0.204	129.79
21	Unprotected	0.0	0.002	8.30
22	Unprotected	0.9	3,103.866	34.62
23	Unprotected	0.6	5.819	10.60
24	Protected	0.8	1.858	5.56
25	Protected	0.1	0.000	26.70
26	Unprotected	0.2	17.326	167.15
27	Unprotected	2.8	4.090	8.95
28	Unprotected	0.5	30.355	13.59
29	Protected	2.0	11.949	1,298.25
30	Protected	3.4	82.591	334.15
31	Unprotected	1.7	5,119.985	155.94
32	Unprotected	0.1	0.037	365.88
33	Unprotected	0.2	0.018	18.05
34	Unprotected	2.9	2.374	8.14
35	Unprotected	0.0	0.011	15.41
36	Protected	0.8	25.733	149.32
37	Unprotected	0.1	0.273	590.54
38	Unprotected	0.0	0.000	16.99
39	Unprotected	0.7	20.110	7.23
40	Protected	0.8	14.030	113.07
41	Unprotected	0.5	9.882	9.42
42	Unprotected	0.0	0.008	12.96
43	Unprotected	0.1	0.828	6.12
44	Unprotected	0.0	0.097	8.80
45	Protected	2.0	155.901	47.34
46	Protected	0.3	2.610	110.42
47	Unprotected	0.1	0.074	26.94
48	Protected	0.0	0.000	10.06
49	Unprotected	0.0	0.006	9.61
50	Unprotected	0.0	0.004	19.74

Table B2.2 – Correlation between pairs of parcels

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
1	1.00	0.38	0.16	-0.21	-0.46	-0.29	-0.64	-0.07	-0.43	0.84	-0.62	-0.27	-0.01	-0.95	0.09	-0.03	0.88	-0.55	-0.45	0.34	-0.42	-0.19	0.16	-0.26	-0.95	-0.73	-0.58	0.01	-0.46	0.80	-0.67	0.95	-0.53	-0.47	-0.38	-0.09	-0.64	-0.43	0.17	0.90	-0.40	-0.05	0.86	-0.48	-0.02	0.46	0.88	-0.14	-0.54	-0.08
2	.	1.00	0.89	0.71	0.52	-1.00	0.38	-0.86	0.55	0.30	0.35	-0.99	0.83	-0.48	0.78	0.81	0.34	0.41	0.51	-0.57	0.53	-0.89	0.87	0.65	-0.57	0.24	0.37	0.77	0.48	-0.19	0.23	0.68	0.36	0.46	0.56	0.72	0.31	0.46	0.94	0.18	0.51	0.74	0.83	0.54	-0.83	-0.48	0.31	-0.88	0.44	0.81
3	.	.	1.00	0.79	0.69	-0.91	0.64	-0.87	0.82	0.06	0.61	-1.00	0.93	-0.24	0.88	0.91	0.06	0.64	0.73	-0.79	0.70	-0.99	0.97	0.76	-0.28	0.52	0.57	0.87	0.76	-0.44	0.52	0.42	0.58	0.67	0.71	0.93	0.51	0.70	0.88	-0.09	0.69	0.82	0.72	0.73	-0.90	-0.71	0.05	-1.00	0.61	0.97
4	.	.	.	1.00	0.85	-0.71	0.88	-0.94	1.00	-0.30	0.87	-0.77	0.86	0.13	0.95	0.95	-0.31	0.82	0.88	-0.91	0.93	-0.90	0.78	0.98	0.09	0.80	0.86	0.87	0.90	-0.79	0.83	0.10	0.78	0.92	0.89	0.87	0.76	0.93	0.89	-0.44	0.99	0.94	0.35	0.79	-0.94	-0.91	-0.36	-0.81	0.89	0.84
5	.	.	.	.	1.00	-0.52	1.00	-0.70	0.86	-0.56	0.85	-0.56	0.86	0.39	0.71	0.84	-0.57	0.94	0.96	-0.84	0.98	-0.65	0.68	0.96	0.38	0.89	0.90	0.74	0.83	-0.88	0.83	-0.20	0.87	0.86	0.90	0.87	0.85	0.90	0.64	-0.67	0.93	0.85	0.07	0.93	-0.84	-0.95	-0.61	-0.78	0.98	0.76
6	.	.	.	.	.	1.00	-0.44	0.93	-0.60	-0.22	-0.48	0.89	-0.87	0.44	-0.95	-0.83	-0.25	-0.58	-0.66	0.67	-0.61	1.00	-0.89	-0.80	0.45	-0.32	-0.47	-0.84	-0.66	0.31	-0.35	-0.65	-0.51	-0.57	-0.67	-0.82	-0.40	-0.59	-0.89	-0.09	-0.66	-0.93	-0.83	-0.56	0.84	0.54	-0.25	0.97	-0.47	-0.89
7	.	.	.	.	.	.	1.00	-0.72	0.86	-0.65	0.90	-0.50	0.67	0.52	0.64	0.69	-0.61	0.96	0.86	-0.93	0.87	-0.55	0.59	0.84	0.54	0.85	0.87	0.65	0.92	-0.93	0.94	-0.32	0.91	0.98	0.97	0.77	0.95	0.98	0.66	-0.77	0.90	0.68	-0.06	0.88	-0.72	-0.87	-0.61	-0.60	0.92	0.74
8	.	.	.	.	.	.	.	1.00	-0.87	0.04	-0.63	0.92	-0.99	0.16	-0.96	-0.91	0.03	-0.71	-0.73	0.80	-0.81	0.96	-0.88	-0.87	0.19	-0.55	-0.64	-0.88	-0.74	0.52	-0.56	-0.35	-0.69	-0.66	-0.85	-0.91	-0.63	-0.73	-1.00	0.17	-0.82	-0.91	-0.61	-0.80	0.96	0.77	0.04	1.00	-0.66	-0.86
9	.	.	.	.	.	.	.	.	1.00	-0.47	1.00	-0.65	0.93	0.35	0.76	0.87	-0.50	0.98	0.92	-0.94	0.89	-0.70	0.77	0.88	0.27	0.92	0.84	0.91	0.91	-0.86	0.92	-0.12	0.86	0.91	0.92	0.89	0.96	0.93	0.69	-0.58	0.91	0.88	0.16	1.00	-0.81	-0.90	-0.50	-0.79	0.88	0.88
10	.	.	.	.	.	.	.	.	.	1.00	-0.62	-0.19	-0.11	-0.88	-0.01	-0.15	1.00	-0.56	-0.60	0.47	-0.48	-0.07	0.05	-0.34	-1.00	-0.71	-0.65	-0.10	-0.57	0.87	-0.77	0.87	-0.70	-0.53	-0.49	-0.20	-0.78	-0.53	0.05	0.94	-0.45	-0.17	0.75	-0.55	0.09	0.59	0.96	-0.04	-0.65	-0.20
11	.	.	.	.	.	.	.	.	.	.	1.00	-0.49	0.79	0.47	0.62	0.69	-0.60	1.00	0.93	-0.94	0.84	-0.60	0.63	0.85	0.47	0.90	0.99	0.77	0.82	-0.92	0.87	-0.32	0.89	0.94	0.92	0.70	0.94	0.89	0.60	-0.68	0.83	0.78	-0.06	0.87	-0.64	-0.86	-0.65	-0.66	0.93	0.72
12	.	.	.	.	.	.	.	.	.	.	.	1.00	-0.88	0.35	-0.87	-0.89	-0.20	-0.61	-0.59	0.66	-0.61	0.97	-0.94	-0.71	0.37	-0.36	-0.51	-0.86	-0.61	0.36	-0.34	-0.52	-0.48	-0.56	-0.63	-0.84	-0.42	-0.58	-0.88	-0.05	-0.61	-0.85	-0.79	-0.64	0.82	0.60	-0.20	0.90	-0.56	-0.78
13	.	.	.	.	.	.	.	.	.	.	.	.	1.00	-0.06	0.83	1.00	-0.10	0.78	0.83	-0.82	0.80	-0.94	0.87	0.90	-0.10	0.66	0.76	0.90	0.80	-0.60	0.67	0.30	0.70	0.81	0.88	0.91	0.60	0.83	0.99	-0.25	0.77	0.96	0.54	0.77	-0.98	-0.78	-0.12	-0.85	0.70	0.88
14	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	-0.16	-0.04	-1.00	0.45	0.35	-0.25	0.35	0.28	-0.25	0.21	0.91	0.61	0.52	-0.08	0.35	-0.72	0.69	-0.89	0.53	0.44	0.38	0.01	0.61	0.41	-0.23	-0.93	0.34	-0.02	-0.85	0.43	0.09	-0.38	-0.90	0.24	0.52	0.00
15	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	0.87	0.00	0.69	0.78	-0.84	0.84	-0.98	1.00	0.84	-0.22	0.55	0.69	0.85	0.68	-0.48	0.56	0.36	0.66	0.66	0.73	0.91	0.61	0.80	0.98	-0.15	0.74	0.91	0.57	0.80	-0.89	-0.72	-0.01	-0.85	0.66	0.88
16	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	-0.12	0.78	0.82	-0.93	0.80	-0.94	0.93	0.96	-0.08	0.64	0.71	0.84	0.86	-0.64	0.62	0.27	0.76	0.75	0.84	1.00	0.62	0.82	0.87	-0.28	0.84	0.87	0.46	0.81	-0.86	-0.75	-0.13	-0.87	0.70	0.97
17	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	-0.60	-0.57	0.41	-0.46	-0.07	0.05	-0.36	-0.93	-0.81	-0.68	-0.11	-0.50	0.80	-0.82	0.87	-0.63	-0.58	-0.50	-0.18	-0.76	-0.55	0.06	0.94	-0.51	-0.16	0.76	-0.59	0.08	0.49	1.00	-0.05	-0.62	-0.16
18	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	0.92	-0.97	0.83	-0.70	0.73	0.85	0.39	0.92	0.93	0.74	0.91	-0.95	0.87	-0.23	0.90	0.87	0.92	0.81	0.96	0.99	0.65	-0.71	0.93	0.86	0.01	0.95	-0.83	-0.85	-0.64	-0.65	0.96	0.79
19	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	-0.89	0.85	-0.78	0.77	0.97	0.35	0.96	0.94	0.78	0.88	-0.82	0.84	-0.18	1.00	0.86	0.86	0.77	0.93	1.00	0.74	-0.64	0.97	0.81	0.11	0.99	-0.87	-0.85	-0.60	-0.81	0.92	0.88
20	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	-0.83	0.81	-0.83	-1.00	-0.25	-0.82	-0.85	-0.86	-1.00	0.87	-0.89	0.05	-0.88	-0.88	-0.93	-0.82	-0.86	-0.90	-0.83	0.61	-0.93	-0.96	-0.24	-0.93	0.93	0.98	0.48	0.77	-0.96	-0.84
21	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	-0.70	0.74	1.00	0.30	0.88	0.87	0.75	0.90	-0.78	0.84	-0.14	0.95	0.85	0.93	0.88	0.80	0.91	0.69	-0.68	0.99	0.79	0.15	1.00	-0.81	-0.95	-0.59	-0.79	0.94	0.94
22	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	-0.99	-0.81	0.27	-0.45	-0.56	-0.89	-0.77	0.46	-0.51	-0.47	-0.57	-0.71	-0.72	-0.96	-0.54	-0.63	-0.89	0.07	-0.75	-0.88	-0.64	-0.70	0.97	0.68	-0.07	0.90	-0.61	-0.92
23	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	0.76	-0.29	0.45	0.59	0.82	0.78	-0.43	0.47	0.44	0.58	0.65	0.75	0.82	0.55	0.68	0.88	-0.09	0.75	0.85	0.63	0.70	-0.92	-0.69	0.05	-0.87	0.62	0.86
24	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	0.18	0.79	0.85	0.97	0.93	-0.81	0.80	0.02	0.93	0.96	0.87	0.90	0.86	0.98	0.77	-0.55	0.94	0.92	0.30	0.98	-0.82	-0.99	-0.39	-0.91	0.89	0.89
25	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	0.62	0.47	-0.13	0.33	-0.73	0.58	-0.91	0.51	0.46	0.31	-0.03	0.54	0.42	-0.26	-0.82	0.32	-0.05	-0.81	0.41	0.12	-0.35	-0.97	0.27	0.52	-0.04
26	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	0.89	0.58	0.92	-1.00	0.90	-0.41	0.96	0.91	0.77	0.71	1.00	0.81	0.49	-0.88	0.82	0.71	-0.23	0.79	-0.60	-0.91	-0.81	-0.47	0.88	0.67
27	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	0.76	0.86	-0.87	0.99	-0.31	0.90	0.99	0.81	0.82	1.00	0.97	0.64	-0.73	0.91	0.76	-0.06	0.88	-0.68	-0.95	-0.66	-0.60	0.83	0.78
28	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	0.80	-0.58	0.64	0.28	0.78	0.77	0.80	0.94	0.65	0.88	0.96	-0.26	0.82	0.85	0.58	0.82	-0.86	-0.83	-0.10	-0.85	0.76	1.00
29	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	-0.77	0.87	-0.16	0.96	0.95	0.97	0.89	0.88	0.94	0.77	-0.64	0.94	0.84	0.10	0.89	-0.87	-0.85	-0.51	-0.71	0.97	0.85
30	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	-0.89	0.52	-0.91	-0.95	-0.79	-0.61	-0.98	-0.91	-0.42	0.90	-0.88	-0.59	0.27	-0.92	0.58	0.89	0.75	0.44	-0.86	-0.58
31	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	-0.45	0.95	0.95	0.85	0.67	0.95	0.95	0.44	-0.80	0.82	0.71	-0.21	0.92	-0.57	-0.93	-0.78	-0.46	0.95	0.65
32	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	-0.33	-0.23	-0.14	0.20	-0.44	-0.19	0.49	0.81	-0.11	0.26	1.00	-0.19	-0.32	0.17	0.82	-0.46	-0.29	0.22
33	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	0.99	0.96	0.75	1.00	0.99	0.62	-0.67	0.83	0.79	-0.05	0.86	-0.68	-0.90	-0.62	-0.56	0.90	0.78
34	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	0.86	0.87	0.94	1.00	0.66	-0.65	0.91	0.84	0.03	0.93	-0.76	-0.92	-0.58	-0.65	0.99	0.84
35	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	0.81	0.85	0.99	0.72	-0.59	1.00	0.77	0.13	0.84	-0.78	-1.00	-0.47	-0.79	0.85	0.80
36	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	0.67	0.89	0.84	-0.33	0.90	0.90	0.44	0.80	-0.86	-0.92	-0.20	-0.95	0.81	0.86
37	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	0.82	0.56	-0.79	0.85	0.72	-0.15	0.90	-0.64	-0.91	-0.73	-0.53	0.83	0.75
38	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	0.74	-0.69	0.87	0.88	0.07	0.98	-0.74	-0.98	-0.58	-0.70	0.99	0.77
39	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	-0.09	0.68	0.91	0.68	0.69	-0.99	-0.76	0.06	-0.98	0.58	0.89
40	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	-0.68	-0.33	0.65	-0.67	0.27	0.74	0.93	0.10	-0.79	-0.33
41	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	0.77	0.17	0.86	-0.82	-0.96	-0.53	-0.78	0.89	0.90
42	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	0.44	0.81	-1.00	-0.83	-0.18	-0.87	0.82	0.96
43	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	0.08	-0.55	-0.10	0.68	-0.69	-0.02	0.48
44	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	-0.85	-0.91	-0.55	-0.71	1.00	0.89
45	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	0.74	0.10	0.86	-0.71	-0.91
46	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	0.59	0.78	-0.97	-0.81
47	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	-0.04	-0.67	-0.19
48	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	-0.62	-0.92
49	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00	0.83
50	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.00

Appendix D: Chapter 5

Appendix D1: List of Features

In this section, we present all $5p^2+106p-50$ features that we used in this study. Note that since $p=3$ in our computational study, the total number of features are 313 features. For convenience, we partition all the proposed features into some subsets and present them next. We use the letter F to represent each subset.

The first subset of features is $F^1 = \{\boldsymbol{c}_1^\intercal \tilde{\boldsymbol{x}}, \boldsymbol{c}_2^\intercal \tilde{\boldsymbol{x}}, \dots, \boldsymbol{c}_p^\intercal \tilde{\boldsymbol{x}}\}$, which will be automatically computed during the pre-ordering process. To incorporate the impact of the size of an instance in the learning process, we introduce $F^2=\{n\}$, $F^3=\{m\}$, and $F^4=\{\text{density}(A)\}$. To incorporate the impact of zero coefficients of the objective functions in the learning process, for each $i \in \{1,\dots,p\}$ we introduce:

$$F_i^5 = \{\text{size}(S_i^5)\},$$

where $S_i^5 = \{c \in \boldsymbol{c}_i : c = 0\}$. To incorporate the impact of positive coefficients of the objective functions in the learning process, for each $i \in \{1,\dots,p\}$ we introduce:

$$F_i^6 = \{\text{size}(S_i^6), \text{Avg}(S_i^6), \text{Min}(S_i^6), \text{Max}(S_i^6), \text{Std}(S_i^6), \text{Median}(S_i^6)\},$$

where $S_i^6 = \{c \in \boldsymbol{c}_i : c > 0\}$. To incorporate the impact of negative coefficients of the objective functions in the learning process, for each $i \in \{1,\dots,p\}$ we introduce:

$$F_i^7 = \{\text{size}(S_i^7), \text{Avg}(S_i^7), \text{Min}(S_i^7), \text{Max}(S_i^7), \text{Std}(S_i^7), \text{Median}(S_i^7)\},$$

where $S_i^7 = \{c \in \boldsymbol{c}_i : c < 0\}$. To establish a relation between the objective functions and A in the learning process, for each $i \in \{1, \dots, p\}$ we introduce:

$$F_i^8 = \{\text{Avg}(S_i^8), \text{Min}(S_i^8), \text{Max}(S_i^8), \text{Std}(S_i^8), \text{Median}(S_i^8)\},$$

where $S_i^8 = \cup_{j \in \{1,\dots,m\}} \{\boldsymbol{c}_i^\intercal \boldsymbol{a}_{j.}^\intercal\}$ and $\boldsymbol{a}_{j.}$ represents row j of matrix A. For the same purpose, for each $i \in \{1, \dots, p\}$ we also introduce:

$$F_i^9 = \boldsymbol{c}_i^\intercal \times A^\intercal \times \boldsymbol{b}.$$

For each $j \in \{1, \dots, m\}$, let $b'_j := b_j + 1$ if $b_j \geq 0$ and $b'_j := b_j - 1$ otherwise. To establish a relation between the positive and negative coefficients in the objective functions and $\boldsymbol{b}$ in the learning process, for each $i \in \{1, \dots, p\}$ and $k \in \{10, 11\}$ we introduce:

$$F_i^k = \{\text{Avg}(S_i^k), \text{Min}(S_i^k), \text{Max}(S_i^k), \text{Std}(S_i^k), \text{Median}(S_i^k)\},$$

where $S_i^{10} = \cup_{j \in \{1,\dots,m\}} \{\frac{\sum_{c \in \boldsymbol{S}_i^6} c}{b'_j}\}$ and $S_i^{11} = \cup_{j \in \{1,\dots,m\}} \{\frac{\sum_{c \in \boldsymbol{S}_i^7} c}{b'_j}\}$.

For each $i \in \{1, \dots, p\}$, let $l_i := \min_{\boldsymbol{x} \in \mathcal{X}_{LR}} \ z_i(\boldsymbol{x})$ and $u_i := \max_{\boldsymbol{x} \in \mathcal{X}_{LR}} \ z_i(\boldsymbol{x})$ where $\mathcal{X}_{LR}$ is the linear programming relaxation of $\mathcal{X}$. To incorporate the impact of the volume of the search region in a projected criterion space, i.e., a $(p-1)$-dimensional criterion space, in the learning process, for each $i \in \{1, \dots, p\}$ we introduce:

$$F_i^{12} = \prod_{j \in \{1,\dots,p\} \setminus \{i\}} (u_j - l_j).$$

Let $\bar{c}'_i := \frac{\sum_{c \in \boldsymbol{c}_i} |c|}{n}$ be the average of the absolute values of the elements in the objective $i \in \{1, \dots, p\}$. Note that we understand that $\boldsymbol{c}_i$ is a vector (and not a set) and hence $c \in \boldsymbol{c}_i$ is not a well-defined mathematical notation. However, for simplicity, we keep this notation as is and basically treat each component of the vector as an element.

We also introduce some features that measures the size of an instance in an indirect way. Specifically, for each $i \in \{1,\dots,p\}$ and $k \in \{13,14,15\}$ we introduce:

$$F_i^k = \{\text{Avg}(S_i^k), \text{Min}(S_i^k), \text{Max}(S_i^k), \text{Std}(S_i^k), \text{Median}(S_i^k)\},$$

to measure n, p and m, respectively, where

$$S_i^{13} := \cup_{j\in\{1,\dots,p\}\setminus\{i\}}\{\frac{\sum_{c\in \boldsymbol{c}_i}|c|}{\bar{c}_j'+1}\},$$

$$S_i^{14} := \cup_{k\in\{1,\dots,n\}}\{\frac{\sum_{j\in\{1,\dots,p\}\setminus\{i\}}|c_{jk}|}{|c_{ik}|+1}\},$$

and

$$S_i^{15} := \cup_{k\in\{1,\dots,n\}}\{\frac{\sum_{j=1}^m |a_{jk}|}{|c_{ik}|+1}\}.$$

Motivated by the idea using the product of two variables for studying the interaction effect between them, for each $i \in \{1,\dots,p\}$, we introduce:

$$F_i^{16} = \{\text{Avg}(S_i^{16}), \text{Min}(S_i^{16}), \text{Max}(S_i^{16}), \text{Std}(S_i^{16}), \text{Median}(S_i^{16})\},$$

where $S_i^{16} = \cup_{j\in\{1,\dots,p\}\setminus\{i\}}\{\sum_{l=1}^n c_{il}c_{jl}\}$. Similarly, we also define a subset of features based on the *leverage score* LS_j of the variable $j \in \{1,\dots,n\}$ in the matrix A. Specifically, for each $i \in \{1\dots,p\}$, we introduce:

$$F_i^{17} = \sum_{j=1}^n c_{ij} LS_j.$$

where $LS_j := \frac{||\boldsymbol{a}_j||^2}{\sum_{l=1}^n ||\boldsymbol{a}_l||^2}$ and $\boldsymbol{a}_j$ represents column j of matrix A for each $j \in \{1,\dots,n\}$. Let $\text{Avg}(C) := \text{Avg}(\boldsymbol{c}_1,\dots,\boldsymbol{c}_p)$, $\text{Std}(C) := \text{Std}(\boldsymbol{c}_1,\dots,\boldsymbol{c}_p)$, and

$$O := \{(-\infty,-1),(-1,-0.5),(-0.5,0),(0,0.5),(0.5,1),(1,\infty)\}.$$

For each $i \in \{1 \ldots, p\}$, we define

$$F_i^{18} = \cup_{(l,u)\in O}\{car(\boldsymbol{c}_i^{l,u})\}$$

where

$$\boldsymbol{c}_i^{l,u} := \{c \in \boldsymbol{c}_i : \mathrm{Avg}(C) + l\ \mathrm{Std}(C) \leq c \leq \mathrm{Avg}(C) + u\ \mathrm{std}(C)\}.$$

The following observation creates the basis of the remaining features.

Observation 3. Let $\alpha_i > 0$ where $i \in \{1, \ldots, p\}$ and β_k where $k \in \{1, \ldots, m\}$ be positive constants. For a MOBLP, its equivalent problem can be constructed as follows:

$$\begin{aligned} &\min\ \{\sum_{j=1}^{n} \alpha_1 c_{1j} x_j, \ldots, \sum_{j=1}^{n} \alpha_p c_{pj} x_j\} && \\ &\text{s.t.}\ \sum_{j=1}^{n} \beta_k a_{kj} x_j \leq \beta_k b_k && \forall\ k \in \{1, \ldots, m\}, \\ &\qquad x_j \in \{0,1\} && \forall\ j \in \{1, \ldots, n\} \end{aligned} \tag{8}$$

Observation 3 is critical because it shows that our ML approach should not be sensitive to positive scaling. So, the remaining features are specifically designed to address this issue. Note that the remaining features are similar to the ones that we have already seen before but they are less sensitive to a positive scaling.

Let $c_i^{max} = \max\{|c_{i1}|, \ldots, |c_{in}|\}$ and $\bar{\boldsymbol{c}}_i = (\frac{c_{i1}}{c_i^{max}}, \ldots, \frac{c_{in}}{c_i^{max}})$. To incorporate the impact of the relative number of zeros, positive, and negative coefficients of the objective functions in the learning process, for each $i \in \{2, \ldots, p\}$, we introduce:

$$F_i^{19} = \{\ln\Big(1 + \frac{car(\bar{\boldsymbol{c}}_1)^0}{1 + car(\bar{\boldsymbol{c}}_i)^0}\Big)\},$$

$$F_i^{20} = \{\ln\Big(1 + \frac{car(\bar{\boldsymbol{c}}_1)^+}{1 + car(\bar{\boldsymbol{c}}_i)^+}\Big)\},$$

and

$$F_i^{21} = \{\ln\Big(1 + \frac{car(\bar{\boldsymbol{c}}_1)^-}{1 + car(\bar{\boldsymbol{c}}_i)^-}\Big)\},$$

where $car(\bar{\boldsymbol{c}}_i)^0$ is the number of elements in $\bar{\boldsymbol{c}}_i$ with zero values. Also, $car(\bar{\boldsymbol{c}}_i)^+$ is the number of elements in $\bar{\boldsymbol{c}}_i$ with positive values. Finally, $car(\bar{\boldsymbol{c}}_i)^-$ is the number of elements in $\bar{\boldsymbol{c}}_i$ with negative values.

The following function is helpful for introducing some other features:

$$g(a) = \left\{ \begin{array}{cc} a+1 & \text{if } \ a \geq 0 \\ a-1 & \text{otherwise} \end{array} \right\}.$$

For each $l \in \{1, \ldots, m\}$, let $a_l^{max} = \max\{|a_{l1}|, \ldots, |a_{ln}|\}$, $\bar{\boldsymbol{a}}_l = (\frac{a_{l1}}{a_l^{max}}, \ldots, \frac{a_{ln}}{a_l^{max}})$ and $\bar{b}_l = \frac{b_l}{a_l^{max}}$. To incorporate the relative impact of the magnitude of objective function coefficients, and constraints in the learning process, for each $i \in \{2, \ldots, p\}$ and $k \in \{22, 23, 24, 25, 26, 27\}$, we introduce:

$$F_i^k = \{\text{Avg}(S_i^k), \text{Min}(S_i^k), \text{Max}(S_i^k), \text{Std}(S_i^k), \text{Median}(S_i^k)\},$$

where

$$S_i^{22} = \{\frac{\bar{c}_{11}}{g(\bar{c}_{i1})}, \ldots, \frac{\bar{c}_{1n}}{g(\bar{c}_{in})}\},$$

$$S_i^{23} = \{\frac{\bar{c}_{11}^2}{g(\bar{c}_{i1}^2)}, \ldots, \frac{\bar{c}_{1n}^2}{g(\bar{c}_{in}^2)}\},$$

$$S_i^{24} = \{\sum_{j=1}^n \frac{\bar{c}_{1j}\bar{a}_{1j}}{ng(\bar{c}_{ij})}, \ldots, \sum_{j=1}^n \frac{\bar{c}_{1j}\bar{a}_{mj}}{ng(\bar{c}_{ij})}\},$$

$$S_i^{25} = \{\sum_{j=1}^n \frac{\bar{c}_{1j}\bar{a}_{1j}}{ng(\bar{c}_{ij})} - \bar{b}_1, \ldots, \sum_{j=1}^n \frac{\bar{c}_{1j}\bar{a}_{mj}}{ng(\bar{c}_{ij})} - \bar{b}_m\},$$

$$S_i^{26} = \{\sum_{j=1}^n \frac{\bar{c}_{1j}^2\bar{a}_{1j}}{ng(\bar{c}_{ij}^2)}, \ldots, \sum_{j=1}^n \frac{\bar{c}_{1j}^2\bar{a}_{mj}}{ng(\bar{c}_{ij}^2)}\},$$

and

$$S_i^{27} = \{\sum_{j=1}^{n} \frac{\bar{c}_{1j}^2 \bar{a}_{1j}}{ng(\bar{c}_{ij}^2)} - \bar{b}_1, \ldots, \sum_{j=1}^{n} \frac{\bar{c}_{1j}^2 \bar{a}_{mj}}{ng(\bar{c}_{ij}^2)} - \bar{b}_m\}.$$

For the same reason, for each $i \in \{1, \ldots, p\}$ and $l \in \{1 \ldots, p\} \backslash \{i\}$, we introduce:

$$F_{il}^{28} = \big\{\text{Avg}(S_{il}^{28}), \text{Min}(S_{il}^{28}), \text{Max}(S_{il}^{28}), \text{Std}(S_{il}^{28}), \text{Median}(S_{il}^{28})\big\},$$

where

$$S_{il}^{28} = \{\frac{\bar{c}_{i1}}{g(\bar{c}_{l1})}, \ldots, \frac{\bar{c}_{in}}{g(\bar{c}_{ln})}\}.$$

Finally, let $\bar{A}_j = \{\bar{a}_{1j}, \ldots, \bar{a}_{mj}\}$ for each $j \in \{1, \ldots, n\}$. For each $i \in \{2, \ldots, p\}$, the following subsets of features are also defined for linking the constraints and objective functions:

$$\begin{aligned} F_i^{29} =&\{\sum_{j=1}^{n} \frac{\bar{c}_{1j}\text{Avg}(\bar{A}_j)}{ng(\bar{c}_{ij})}, \sum_{j=1}^{n} \frac{\bar{c}_{1j}\text{Min}(\bar{A}_j)}{ng(\bar{c}_{ij})}, \sum_{j=1}^{n} \frac{\bar{c}_{1j}\text{Max}(\bar{A}_j)}{ng(\bar{c}_{ij})}, \sum_{j=1}^{n} \frac{\bar{c}_{1j}\text{Std}(\bar{A}_j)}{ng(\bar{c}_{ij})}, \\ &\sum_{j=1}^{n} \frac{\bar{c}_{1j}\text{Median}(\bar{A}_j)}{ng(\bar{c}_{ij})}\}, \end{aligned}$$

$$\begin{aligned} F_i^{30} =&\{\sum_{j=1}^{n} \frac{\bar{c}_{1j}\text{Avg}(\bar{A}_j)}{ng(\bar{c}_{ij})} - \text{Avg}(\bar{b}_j), \sum_{j=1}^{n} \frac{\bar{c}_{1j}\text{Min}(\bar{A}_j)}{ng(\bar{c}_{ij})} - \text{Min}(\bar{b}_j), \sum_{j=1}^{n} \frac{\bar{c}_{1j}\text{Max}(\bar{A}_j)}{ng(\bar{c}_{ij})} - \text{Max}(\bar{b}_j), \\ &\sum_{j=1}^{n} \frac{\bar{c}_{1j}\text{Std}(\bar{A}_j)}{ng(\bar{c}_{ij})} - \text{Std}(\bar{b}_j), \sum_{j=1}^{n} \frac{\bar{c}_{1j}\text{Median}(\bar{A}_j)}{ng(\bar{c}_{ij})} - \text{Median}(\bar{b}_j)\}, \end{aligned}$$

$$\begin{aligned} F_i^{31} =&\{\sum_{j=1}^{n} \frac{\bar{c}_{1j}^2\text{Avg}(\bar{A}_j)}{ng(\bar{c}_{ij}^2)}, \sum_{j=1}^{n} \frac{\bar{c}_{1j}^2\text{Min}(\bar{A}_j)}{ng(\bar{c}_{ij}^2)}, \sum_{j=1}^{n} \frac{\bar{c}_{1j}^2\text{Max}(\bar{A}_j)}{ng(\bar{c}_{ij}^2)}, \sum_{j=1}^{n} \frac{\bar{c}_{1j}^2\text{Std}(\bar{A}_j)}{ng(\bar{c}_{ij}^2)}, \\ &\sum_{j=1}^{n} \frac{\bar{c}_{1j}^{@}\text{Median}(\bar{A}_j)}{ng(\bar{c}_{ij}^2)}\}, \end{aligned}$$

$$\begin{aligned}F_i^{32} =&\{\sum_{j=1}^{n}\frac{\bar{c}_{1j}^2\mathrm{Avg}(\bar{A}_j)}{ng(\bar{c}_{ij}^2)} - \mathrm{Avg}(\bar{b}_j), \sum_{j=1}^{n}\frac{\bar{c}_{1j}^2\mathrm{Min}(\bar{A}_j)}{ng(\bar{c}_{ij}^2)} - \mathrm{Min}(\bar{b}_j), \sum_{j=1}^{n}\frac{\bar{c}_{1j}^2\mathrm{Max}(\bar{A}_j)}{ng(\bar{c}_{ij}^2)} - \mathrm{Max}(\bar{b}_j),\\ &\sum_{j=1}^{n}\frac{\bar{c}_{1j}^2\mathrm{Std}(\bar{A}_j)}{ng(\bar{c}_{ij}^2)} - \mathrm{Std}(\bar{b}_j), \sum_{j=1}^{n}\frac{\bar{c}_{1j}^2\mathrm{Median}(\bar{A}_j)}{ng(\bar{c}_{ij}^2)} - \mathrm{Median}(\bar{b}_j)\},\end{aligned}$$

where $\bar{c}_{ij}^2 = \bar{c}_{ij}\bar{c}_{ij}$ for each $i \in \{1, \ldots, p\}$ and $j \in \{1 \ldots, n\}$.