Solution algorithms for multi-objective integer linear programming models

– A study of the branch-and-bound algorithm applied to the multi-objective case

A PhD Dissertation

by

Nicolas Forget

Aarhus BSS, Aarhus University

Department of Economics and Business Economics

August 2022

Solution algorithms for multi-objective integer linear programming models

– A study of the branch-and-bound algorithm applied to the multi-objective case

Thesis advisors

– Lars Relund Nielsen

– Sune Lauth Gadegaard

Nicolas Forget
Department of Economics and Business Economics
School of Business and Social Sciences
Aarhus University
Fuglesangs Allé 4
DK-8210
Denmark
E-mail:nforget@econ.au.dk

The LaTeX template used to typeset this document is originally created by Sune Lauth Gadegaard and adapted by Nicolas Forget
The template was originally designed for the PhD thesis Gadegaard (2016)
Font: Latin modern. 12pt scaled to 95 percent
Produced with pdfLaTeX and the *memoir*–class

Table of contents

Summary **v**

Abstract **ix**

Resumé **xi**

Acknowledgments **xiii**

Notation **xv**

1 Introduction **1**

1.1 Multi-objective optimization . . . 3

1.1.1 Bound sets . . . 6

1.1.2 Solution methods for MOILP . . . 9

1.2 Objective Space Search algorithms . . . 9

1.3 Decision Space Search algorithms . . . 12

1.3.1 Node selection . . . 13

1.3.2 Lower bound set computation . . . 15

1.3.3 Fathoming rules . . . 17

1.3.4 Creation of sub-problems . . . 20

1.3.5 Termination of the algorithm . . . 21

1.4 Contributions and structure of the dissertation . . . 24

2 Warm-starting lower bound set computations for branch-and-bound algorithms for multi objective integer linear programs **25**

2.1 Introduction . . . 26

2.2 Preliminaries . . . 30

2.2.1 Polyhedral theory . . . 30

2.2.2 Bound sets . . . 31

2.3 A branch-and-bound framework for MOILP . . . 32
2.4 Linear relaxation for MOBB . . . 34
2.4.1 Warm-starting Benson-like algorithms in MOBB . . . 37
2.5 Computational experiments . . . 39
2.5.1 Implementation details and algorithm configurations . . . 40
2.5.2 Test instances . . . 42
2.5.3 Performance of the different algorithm configurations . . . 44
2.5.4 Variable selection - Rules for choosing the bound . . . 44
2.5.5 Detailed performance of different algorithm parts . . . 47
2.5.6 Fathoming nodes . . . 48
2.5.7 Geometric properties of the lower bound set during the algorithm . . 49
2.5.8 Proving optimality . . . 51
2.5.9 Performance of the MOBB algorithm compared to an objective space search algorithm . . . 52
2.6 Conclusion . . . 55
2.A Notes on the the unbounded case . . . 57
2.B Problem classes . . . 58
2.B.1 Production Planning Problem . . . 58
2.B.2 Uncapacitated Facility Location Problem . . . 58

3 Branch-and-bound and objective branching with three or more objectives 61
3.1 Introduction . . . 62
3.2 Definitions and notations . . . 66
3.2.1 Bound sets . . . 67
3.2.2 Search region and local upper bounds . . . 67
3.3 General multi-objective Branch-and-Bound framework . . . 69
3.4 Objective branching . . . 72
3.4.1 Complications of going from two to three objectives . . . 74
3.4.2 Objective branching in the multi-objective case . . . 76
Merging operations on local upper bounds . . . 77
Desirable properties of the set of super local upper bounds . . . 78
An algorithm to compute a set of super local upper bounds . . . 79
Implications of Property 3.2 . . . 80
3.4.3 An alternative branching strategy using an upper bound on the objectives . . . 83
3.5 Computational experiments . . . 83
3.5.1 Test instances . . . 85
3.5.2 Performance of the different algorithm configurations . . . 87

3.5.3 Objective branching: a closer look 88
3.5.4 Branching tree structure when using `fullOB` 91
3.5.5 Comparison with an Objective Space Search algorithm 93
3.6 Conclusion 95

4 Enhancing branch-and-bound for multi-objective 0-1 programming 97
4.1 Introduction 98
4.2 Definitions and notation 100
4.3 Related work 101
4.4 Branch-and-bound framework 103
4.5 Enhanced objective branching 105
4.5.1 Probing 107
A naive strategy 107
An advanced strategy 109
Combining probing and bounding 110
4.5.2 Objective branching based cover inequalities 110
4.6 Node selection rules 112
4.6.1 Weighted-sum rule 112
4.6.2 Gap measure rule 114
4.7 Experiments 115
4.7.1 Probing and objective branching 117
4.7.2 Combining probing and bounding 120
4.7.3 Node selection rules 121
4.7.4 Objective branching and cover cuts inequalities 122
4.7.5 Investigating variable selection rules: the case of CFLP 123
4.7.6 Enumeration in MOBB 124
4.7.7 Comparison with objective space search algorithms 125
4.8 Conclusion 126

Bibliography 129

Summary

Many real-world optimization problems can be expressed as a linear program with integer variables, e.g. in logistics, scheduling, supply chain management, ect. Usually, an objective is defined by the decision maker (minimize a cost, a distance, a travelling time, maximize a profit, ect.), and the problem is optimized accordingly. However, considering a unique objective function does not necessarily lead to a solution that satisfies the decision maker. To overcome that, multiple objectives can be considered simultaneously, and solutions that are good trade-offs between these objectives are provided instead. Multi-objective optimization is the science of computing these interesting trade-offs.

A popular algorithm used to solve such optimization problems is the branch-and-bound algorithm. It has received a lot of attention and improvements over the paste decades for solving problems with a single objective, and nowadays, even large problems can be solved using this method. Extensions to multiple objectives have been proposed in the past, but it is only recently that the branch-and-bound algorithm has been able to efficiently solve problems with two objective functions.

Given the recent success of the method for the bi-objective case, the main purpose of this thesis is to develop a branch-and-bound framework that can solve problems with three or more objective functions efficiently. Indeed, while significant difficulties occur when a second objective is considered, additional challenges arise when a third objective is introduced. Thus, the goal here is to identify and overcome these challenges, study the performance and the behavior of the branch-and-bound algorithm on such problems, and propose improvements that lead to an efficient framework. All chapters of this thesis are directed towards this goal, and evidence that the resulting framework obtain significant speed-ups using the proposed approaches is given through the dissertation. In the first chapter, research gaps and potential directions for improvement are identified, whereas in the three other chapters, various key components of the algorithm are enhanced. The results of each chapter are cumulative, i.e. chapter four relies on results from chapter three, that benefits itself from results presented in chapter two. Together, they build an efficient branch-and-bound framework for solving linear optimization problems with integer variables and three or more objective functions.

Note that the chapters are written in a way such that they can be read independently.

In the first chapter, referred to as *Introduction*, a detailed description of the problem considered is provided as well as definitions and notations used through the dissertation. Then, an exhaustive literature review is given with a special focus on the branch-and-bound algorithm. In particular, research gaps are identified, key components of efficient bi-objective frameworks are highlighted, and research directions for the thesis are stated.

The second chapter, *Warm-starting lower bound set computations for branch-and-bound algorithms for multi objective integer linear programs*, lays the foundation of a branch-and-bound framework that can solve problems with three or more objective functions. Besides handling any number of objectives, the novelty lies in the use of the linear relaxation to generate lower bound sets and in its computation, which is warm-started using particular properties of the algorithm. Extensive experiments are carried out on a set of various problem classes presenting different properties and evidence that the proposed warm-starting procedure results in a significant speed-up is reported. Moreover, statistics on the behavior of the algorithm are presented and analyzed with a particular focus on the lower bound sets.

In the third chapter, *Branch-and-bound and objective branching with three or more objectives*, the goal is to extend objective branching to the case where three or more objective functions are considered. Objective branching consists of reducing the search region by creating sub-problems in a particular way with the expectation that the overall computational costs decrease due to the smaller size of the problems being handled. Objective branching is one of the key features that significantly improved the branch-and-bound algorithm in the bi-objective case but, unfortunately, its extension to any number of objectives is not straightforward. This paper identifies the difficulties that arise in this case and proposes a method to compute objective branching in a way that satisfies a number of desirable properties. Furthermore, an alternative to objective branching that relies on a similar principle is also proposed. Experiments are carried out on five different problem classes with three, four, and five objective functions. The results show that improvements in terms of CPU time are observed most of the time, even though the numbers are not as promising as in the bi-objective case due to the difficulties occurring with a higher number of objectives. Similar to Chapter 2, statistics on the behavior of the algorithm are analyzed with a particular focus on branching.

Finally, the fourth chapter, *Enhancing branch-and-bound for multi-objective 0-1 programming*, restricts the analysis to problems with binary variables only. However, with a few changes, all the ideas presented can be applied to problems with integer variables. The chapter can be divided into three main contributions. First, an extension of probing to the multi-objective case is proposed. Probing is a technique used in the single-objective case that consists of fixing variables to 0 or 1, and derive information from the consequences observed. It has been proven to be a very efficient approach in the single-objective case, and the experiments carried out in this chapter show that when coupled with objective branching, significant speed-ups are obtained in the multi-objective case. Then, different node selection strategies based on the best-bound principle are explored. These are tested against the traditional depth-first and breadth-first strategies, and the experiments show that some of the proposed rules result in lower CPU times. Finally, small experiments are carried out on other features, namely problem specific variable selection rules, cut generation on objective branching constraints, and pure enumeration of solutions, i.e. without lower bound computation. The results show that some of these elements can generate a speed-up too.

In each chapter, the new branch-and-bound framework proposed is tested against other algorithms from the literature, and the final framework appears to be competitive on some problem classes.

Abstract

In this thesis, the aim is to design an efficient branch-and-bound algorithm to solve linear optimization problems with integer variables and three or more objective functions.

The first chapter provides an exhaustive overview of the state-of-the-art with a particular focus on branch-and-bound methods. Research gaps are identified and the contributions of the thesis are presented.

In the second chapter, the basic branch-and-bound framework, that can be considered as the skeleton of this thesis, is built. It can handle any number of objectives and uses the linear relaxation as lower bound sets. Moreover, the computation of the latter is improved by a warm-starting procedure developed in this chapter, and experiments show that smaller CPU times are reached by using the proposed approach.

The third chapter is dedicated to the extension of objective branching, an effective method to create sub-problems in the bi-objective case, to any number of objective functions. A number of difficulties are presented, and an algorithm satisfying a set of properties is proposed to compute the sub-problems in the given context. The experiments show that better CPU times are achieved most of the time.

The fourth chapter further improves objective branching by introducing probing at the creation of the sub-problems, resulting in significant speed-ups. Besides, various node selection rules are tested and found to be more effective than those usually used in the literature. Finally, other features such as cut generation, variable selection rules, and enumeration techniques are explored. The resulting branch-and-bound framework is competitive against other recent methods from the literature.

Resumé

Denne afhandling har til formål at designe en effektiv branch-and-bound algoritme til at løse lineære optimeringsproblemer med heltalsvariabler og tre eller flere objektfunktioner.

Det første kapitel giver et udtømmende overblik over litteraturen med særligt fokus på branch-and-bound metoder. Forskningshuller identificeres, og afhandlingens bidrag præsenteres.

I det andet kapitel opbygges den grundlæggende branch-and-bound tilgang, der kan betragtes som skelettet i denne afhandling. Den kan håndtere et hvilket som helst antal objektfunktioner og bruger den lineære relaksering som nedre grænse. Desuden er beregningen af sidstnævnte forbedret med en warm-starting procedure udviklet i samme kapitel, og eksperimenter viser, at bedre CPU-tider opnås ved at bruge den foreslåede fremgangsmåde.

Det tredje kapitel er dedikeret til udvidelsen af branching i kriterierummet (en effektiv metode til at skabe delproblemer i tilfældet med to objektfunktioner) til et arbitrært antal objektfunktioner. En række udfordringer præsenteres, og en algoritme, der opfylder en mængde af egenskaber, foreslås til at skabe delproblemerne i proceduren. De numeriske beregninger viser, at der opnås bedre CPU-tider det meste af tiden.

Det fjerde kapitel forbedrer yderligere branching i kriterierummet ved at introducere såkaldt probing i forbindelse med oprettelsen af delproblemet, hvilket resulterer i betydelige reduktioner i CPU tiden. Desuden testes forskellige regler til udvælgelse af nye delproblemer og disse viser sig at være mere effektive end dem, der oftest bruges i litteraturen. Til sidst udforskes andre tiltag såsom snitplansgenerering, regler for udvælgelse af branching variabel, og enumerationsteknikker. Den resulterende branch-and-bound algoritme er konkurrencedygtig i forhold til de bedste metoder fra litteraturen.

Acknowledgments

My PhD started at the Economics and Business Economics department of Aarhus BSS in September 2019. These three years have been a valuable and stimulating journey. I have discovered many things, improved the knowledge accumulated through my studies, developed new skills, and learned a lot about myself. On top of that, I have had the opportunity to meet a lot of supportive and inspiring people, whom I would like to thank here.

First, I would like to thank my supervisor Lars Relund Nielsen and my co-supervisor Sune Lauth Gadegaard. Both of them have been very welcoming when I first arrived and have always been very supportive since then, be it for the academic or personal matters. I thank them for the nice research discussions and guidance they offered me. I have learned so much from them and I am very grateful for all the knowledge and help they have given me in these three years.

Thank you to Kathrin Klamroth from Wuppertal in Germany and Athony Przybylski from Nantes in France, who accepted to be co-author on one of the chapters of this thesis. Despite the distance, they always found the time to share nice research discussions and provide feedback to my work. A particular thank to Kathrin Klamroth for welcoming me and Sune as guests in Wuppertal for a few days in December 2019.

In September 2021, I had the privilege to visit Sophie Parragh at Johannes Kepler University in Linz, Austria. Thank you for the opportunity, for the great academic discussions, and for the contributions to my thesis. I would like to express my special thank for the very warm welcome to all members of the Institut für Produktions- und Logistikmanagement, and to Sabine Frank, who helped me with the practical matters. Finally, I would like to thank again Sophie Parragh, who invited me a second time in Linz for a three week visit in April 2022.

Thank you to all my colleagues at the CORAL (Cluster for Operation Research, Analytics, and Logistics) for the relaxed conversations over lunch breaks and interesting seminars. Thank you to my fellow PhD students at the CORAL Lone Kiilerich Christensen, Maximiliano Cubillos, Quiping Ma, Johan Clausen, Farnaz Farzadnia, Shohre Zehtabian, Julian Baals, and Jesper Bang Mikkelsen, for the valuable experience sharing, the delicious cakes, and all the great times together. I would like to express a special thank to Johan Clausen, who shared an office with me during these three years. Thank you for all the fun, long and nice discussions we have had, and for helping me many times with practical and personal matters here in Denmark.

I would like to thank Karin Vinding, Solveig Nygaard Sørensen, and Lene Engelst Christensen for proofreading my chapters, and Betina Sørensen, Solveig Nygaard Sørensen, and Charlotte Sparrevohn for their help with administrative matters.

I am grateful to Michael Stiglmayr (University of Wuppertal, Germany), Serpil Sayın (Koç University, Turkey), and Michael Malmros Sørensen (Aarhus University, Denmark), the members of the PhD committee, who kindly accepted to spend time reading carefully this manuscript, and to provide valuable feedback on my work.

Finally, I would like to express my gratitude to my family, who supported me from the start and always reached out when I needed. To my parents Sophie Pavageau and Jean Pavageau, to my sister Marion Forget, and my extended family, thank you. To Cyrielle Cadio, Harmony Agasse, Jeremy Gaulier, Corentin Bouvier, and all my friends from France, who have always been there for me and helped me go through the pandemic despite the distance, thank you.

Nicolas Forget
August 2022

Notation

Throughout this dissertation I have aimed at keeping a consistent use of symbols. In general, calligraphic capitals denote sets, upper case Latin letters denote problems, and lower case Latin letters denote elements of sets, variables, functions, parameters, or indices. Table 1 summarizes the standardized notation for the dissertation.

Notation	Explanation of notation
n	Number of variables.
p	Number of objectives.
y	A point in the objective space $\mathbb{R}^p$.
x	A feasible solution.
z	Objective function.
$z(x)$	Objective vector of a feasible solution.
$\mathcal{X}$	Feasible set of an optimization problem.
$\mathcal{Y}$	Feasible set in objective space of a multi–objective optimization problem.
$y^1 < y^2$	$y_k^1 < y_k^2$, for all $k \in \{1, ..., p\}$.
$y^1 \leqq y^2$	$y_k^1 \leq y_k^2$, for all $k \in \{1, ..., p\}$.
$y^1 \leq y^2$	$y^1 \leqq y^2$ and $y^1 \neq y^2$.
$\mathbb{R}^p_>$	$\{y \in \mathbb{R}^p \mid y > 0\}$.
$\mathbb{R}^p_\geqq$	$\{y \in \mathbb{R}^p \mid y \geqq 0\}$.
$\mathbb{R}^p_\geq$	$\{y \in \mathbb{R}^p \mid y \geq 0\}$.
S_N	$\{s \in \mathcal{S} \mid \nexists s' \in \mathcal{S},\ s' \leqslant s\}$. The non–dominated set of $S \subset \mathbb{R}^p$.
$\mathcal{Y}_N$	Set of non–dominated points of a multi–objective optimization problem.
$\mathcal{U}$	Upper bound set on the non–dominated set of a multi–objective optimization problem.
$\mathcal{N}(\mathcal{U})$	Set of local upper bounds corresponding to the upper bound set $\mathcal{U}$.
u	A local upper bound.
η	Node of the branch-and-bound tree.
$\mathcal{L}(\eta)$	Lower bound set on the non–dominated set of the problem contained in node η.

Table 1: Notation

First Chapter

Introduction

History: This chapter is the basis for a future review paper. It has been adapted to be the introduction of the thesis.

Introduction

Nicolas Forget*

* Department of Economics and Business Economics, School of Business and Social Sciences, Aarhus University, Denmark

In the real world, many decisions can be formulated as an optimization problem. For instance, a decision maker wants to minimize a cost or to maximize a profit, under certain constraints. Often, a single, unique objective is optimized. However, the nature of some problems implies the need to include more objectives that take into account several incomparable measures. For example, when a city examines where to build firefighter stations, multiple objectives naturally emerge. First, a city-planner could aim at minimizing the average distance between a firefighter station and the citizens. This ensures that the solution is effective, i.e. citizens are in general close to a firefighter station and thus can receive assistance in a reasonable amount of time. However, this is not necessarily a desirable situation. Indeed, if the city has a very dense city center but only a few individuals that live further away in the outskirts of the city, the stations will most likely be located very close to the city center. In this case, if an emergency occurs in one of the households in the periphery areas, the firefighters may not be able to arrive fast enough. Hence, the city may as well be interested in minimizing the maximal distance to a station. That way, fairness among the citizens is also considered.

It is easy to see that these two measures are conflicting, and the city is interested in a good trade-off between these two objectives, i.e. a solution such that there is no other solution that is both more effective and fair. Multi-objective optimization is the science of finding all or some of the interesting compromises to such problems. Considering two objectives at the same time is a special case of multi-objective optimization and is called bi-objective optimization. However, many additional objectives can be added to the problem.

In recent years, more and more real world problems consider multiple objectives. For instance, with the rise of new environmental challenges and new measures regarding climate regularly introduced by governments, a company may be interested in minimizing e.g. its CO_2 emissions, waste production, or carbon footprint in addition to the standard objectives.

This paper focuses on a particular class of problem that considers linear objective functions, linear constraints, and integer variables only. A complete overview of the existing methods to solve such problems is provided. The various techniques are divided into two categories here. Section 1.1 is dedicated to the presentation of the problem and introduces important notations and definitions. In Section 1.2, a brief state-of-the-art of the first class of algorithms is given, whereas Section 1.3 explores the state-of-the-art of the second class of algorithms. The latter being the focus of this dissertation, a deep and exhaustive analysis is provided, and research gaps are identified. Finally, Section 1.4 will present the research gaps that are addressed in this dissertation.

1.1 Multi-objective optimization

A *Multi-Objective Linear Program* (MOLP) with p objective functions, n decision variables, and m constraints can be written in the following form:

$$\begin{aligned} \min\ & z(x) = Cx \\ \text{s.t. } & Ax \geqq b \\ & x \in \mathcal{D} \end{aligned}$$

where C is a $p \times n$ matrix representing the coefficients of the objective functions. The objective functions are written as $z(x) = Cx$. The matrix A is an $m \times n$ matrix representing the coefficients of the constraints, and b is a vector of size m representing the right-hand side of the constraints. Finally, the set $\mathcal{D} \subseteq \mathbb{R}^n$ corresponds to the domain of the variables. If $\mathcal{D} = \mathbb{R}^n_+$, the problem is a *Multi-Objective Continuous Linear Program* (MOCLP); if $\mathcal{D} = \mathbb{Z}^n$, the problem is a *Multi-Objective Integer Linear Program* (MOILP); and if $\mathcal{D} = \{0,1\}^n$, the problem becomes a *Multi-Objective 0-1 Linear Program* (MO01LP). A MO01LP with a specific structure in the constraints is also sometimes called a *Multi-Objective Combinatorial Optimization* problem. Finally, there exists problems that contain both continuous and integer variables. Let $0 < n_B < n$ be the number of integer variables, if $\mathcal{D} = \mathbb{Z}^{n_B} \times \mathbb{R}^{n-n_B}_+$, the problem is a *Multi-Objective Mixed Integer Linear Program* (MOMILP); whereas if $\mathcal{D} = \{0,1\}^{n_B} \times \mathbb{R}^{n-n_B}_+$, the problem is a *Multi-Objective Mixed 0-1 Linear Program* (MOM01LP).

A vector of decision variables $x \in \mathbb{R}^n$ is called a *solution*. As a result, the set $\mathbb{R}^n$ is also named *decision space* or solution space, and the *feasible set* $\mathcal{X} = \{x \in \mathcal{D} : Ax \geqq b\} \subset \mathbb{R}^n$ is the set of all solutions that satisfies the constraints of the MOLP at hand. A solution x such that $x \in \mathcal{X}$ is called a *feasible solution*.

A solution $x \in \mathbb{R}^n$ can be mapped through the objective function, resulting in a vector $y = Cx \in \mathbb{R}^p$. The set $\mathbb{R}^p$ is named the *objective space*, and a vector $y \in \mathbb{R}^p$ is called a *point* or an *objective vector*. Each feasible solution $x \in \mathcal{X}$ can be mapped through the p objective functions, resulting in the *set of feasible points* $\mathcal{Y} := C\mathcal{X} = \{y \in \mathbb{R}^p : y = Cx,\ x \in \mathcal{X}\}$. A point $y \in \mathcal{Y}$ is called a *feasible point*.

Contrary to the single-objective case where each solution is evaluated by a unique value, namely the value of the objective function, solutions are evaluated by a vector of objective function values in the multi-objective case. This implies that comparisons between solutions involve comparisons between vectors, and new relations need to be defined to accomplish that. For this purpose, we introduce the dominance relations in Definition 1.1 and illustrate them in Figure 1.1a.

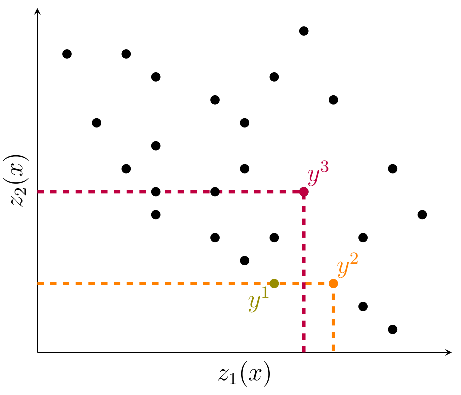
Definition 1.1. Let $y^1, y^2 \in \mathbb{R}^p$. We have the three following dominance relations:

- *weak dominance*: $y^1 \leqq y^2$ if $y^1_k \leq y^2_k$ for each $k \in \{1, ..., p\}$. We say that y^1 weakly dominates y^2;
- *dominance*: $y^1 \leqslant y^2$ if $y^1 \leqq y^2$ and $y^1 \neq y^2$. We say that y^1 dominates y^2;
- *strict dominance*: $y^1 < y^2$ if $y^1_k < y^2_k$ for each $k \in \{1, ..., p\}$. We say that y^1 strictly dominates y^2.

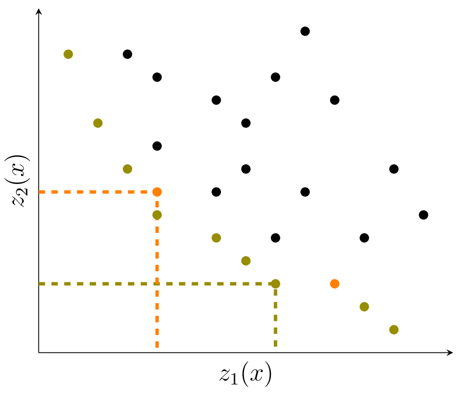
In Figure 1.1a, we can see that y^1 dominates both y^2 and y^3, since y^1 is better or equal on both objectives and strictly better in at least one of them. Moreover, y^1 strictly dominates y^3 as it is strictly better in all objectives, but does not strictly dominate y^2 since they are equal in the second objective. Finally, nothing can be concluded between y^2 and y^3 with respect to the dominance relations from Definition 1.1 since they are both better than the other in one objective, but worse on the remaining objective.

When multiple conflicting objectives are considered, there is usually no feasible solution that minimizes all the objectives at the same time. Instead, the focus is on finding feasible solutions that represent good trade-offs between the objectives. A classical way to define what a good trade-off solution is is to use the dominance relations. Let $x \in \mathcal{X}$, we say that x is *efficient* if there is no other feasible solution $x' \in \mathcal{X}$ such that $x \neq x'$ and $z(x') \leqslant z(x)$, and we call its image in the objective space $z(x)$ a *non-dominated point.* From there, we can define the *set of efficient solutions* $\mathcal{X}_E = \{x \in \mathcal{X} : \nexists x' \in \mathcal{X}, z(x') \leqslant z(x)\}$ and the *set of non-dominated points* $\mathcal{Y}_N := C\mathcal{X}_E = \{y \in \mathcal{Y} : \nexists y' \in \mathcal{Y}, y' \leqslant y\}$. It is important to note that since multiple efficient solutions can have similar images, there can exist one or multiple sub-sets $\mathcal{X}'_E \subset \mathcal{X}_E$ such that $C\mathcal{X}'_E = \mathcal{Y}_N$. It is said that a *complete set* of efficient solutions is computed if at least one solution for each non-dominated point is obtained, i.e. $\mathcal{Y}_N$ is computed. On the other hand, a *maximal complete set* of efficient solutions is computed if, for each non-dominated point, all the corresponding efficient solutions are obtained, in which case $\mathcal{X}_E$ is computed. In the example from Figure 1.1, the set of non-dominated points is depicted in green in Figure 1.1b.

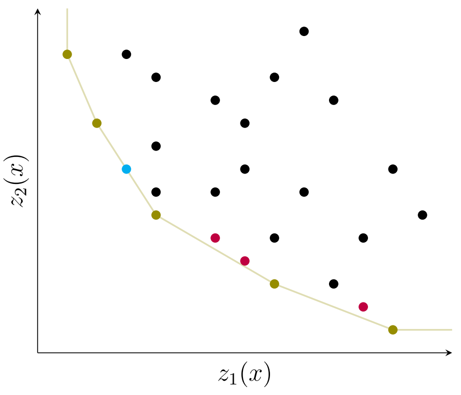
Analogously to $\mathcal{X}_E$ and $\mathcal{Y}_N$, we define the *set of weakly efficient solutions* as $\mathcal{X}_{WE} = \{x \in \mathcal{X} : \nexists x' \in \mathcal{X}, z(x') < z(x)\}$ and the *set of weakly non-dominated points* as $\mathcal{Y}_{WN} = \{y \in \mathcal{Y} : \nexists y' \in \mathcal{Y}, y' < y\}$. Any solution $x \in \mathcal{X}_{WE}$ is called *weakly efficient* and a point $y \in \mathcal{Y}_{WN}$ is said to be *weakly non-dominated.* It is interesting to note that, necessarily, $\mathcal{Y}_N \subseteq \mathcal{Y}_{WN}$. In Figure 1.1b, $\mathcal{Y}_{WN}$ is given by the green and orange points, the latter corresponding to points that are weakly non-dominated but not non-dominated.



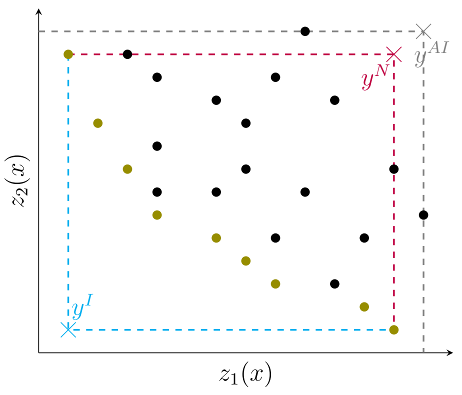
(a) The point y^1 strictly dominates y^3 since it is strictly lower on both objectives. The point y^1 dominates y^2 but does not strictly dominate it since they are equal on objective 2. Finally, nothing can be concluded regarding the relation between y^2 and y^3 as they are both better on one objective but worse on the other.



(b) In green is depicted $\mathcal{Y}_N$, the set of non-dominated points. The green and orange points correspond to the set of weakly non-dominated points $\mathcal{Y}_{WN}$. In particular, the orange points are those that are weakly non-dominated but not non-dominated.



(c) The convex hull of the non-dominated points is given by the green line. The supported extreme points corresponds to the green points, and the supported non-extreme to the blue points. The red points represent the unsupported points.



(d) The set of non-dominated points $\mathcal{Y}_N$ is given by the green points. The ideal point y^I corresponds to the blue cross. The nadir point y^N is represented by the red cross. Finally, the anti-ideal point y^{AI} is depicted by the gray cross.

Figure 1.1: An example of a set of feasible points of a MOILP with two objective functions in minimization, plotted in the objective space. Each point corresponds to a feasible point, and each axis corresponds to one objective function.

The non-dominated points can be further classified into three categories, namely supported extreme, supported non-extreme, and unsupported points. First, we define $\mathbb{R}^p_{\geqq} := \{z \in \mathbb{R}^p : z \geqq 0\}$, $\mathbb{R}^p_{>} := \{z \in \mathbb{R}^p : z > 0\}$, and $\mathbb{R}^p_{\geqslant} := \{z \in \mathbb{R}^p : z \geqslant 0\}$. Let conv(.) define the convex hull operator, a *supported* point is a non-dominated point that lies on the boundary of $\text{conv}(\mathcal{Y}_N) + \mathbb{R}^p_{\geqq}$. In Figure 1.1c, the boundary of $\text{conv}(\mathcal{Y}_N) + \mathbb{R}^p_{\geqq}$ is depicted by the green line, and the supported points are drawn in green and blue. A supported point that is an extreme point of $\text{conv}(\mathcal{Y}_N) + \mathbb{R}^p_{\geqq}$ is called *supported extreme*, and is named *supported non-extreme* otherwise. In Figure 1.1c, supported extreme points are given in green, and supported non-extreme points are depicted in blue. Finally, a non-dominated point that does not lie on the boundary but in the interior of the convex hull is called *non-supported* and shown in red in Figure 1.1c.

1.1.1 Bound sets

Three particular points are worth mentioning because they are often used in multi-objective optimization. First, the *ideal point* y^I is the point that minimizes all the objectives among all feasible points at the same time, i.e. $y^I_k = \min_{y\in\mathcal{Y}} y_k$. It is interesting to note that $y^I_k = \min_{y\in\mathcal{Y}_N} y_k$ also holds true, i.e. that it is equivalent to say that it minimizes all the objectives among the non-dominated points. Due to the conflicting nature of the objectives, the ideal point is often not feasible, i.e. usually $y^I \notin \mathcal{Y}$. On the contrary, the *anti-ideal point* is the point that maximizes all the objectives among the feasible points at the same time, i.e. $y^{AI}_k = \max_{y\in\mathcal{Y}} y_k$. Finally, the *nadir point* y^N is the point that maximizes all the objectives, but considering non-dominated points only, i.e. $y^N_k = \max_{y\in\mathcal{Y}_N} y_k$. The ideal, anti-ideal, and nadir points are depicted in blue, gray, and red respectively in Figure 1.1d.

In the single-objective case, the optimal value z^* is often estimated using lower bounds $l \in \mathbb{R}$ and upper bounds $u \in \mathbb{R}$, i.e. $l \leq z^* \leq u$. In the multi-objective case, since there is not a unique optimal value but a set of non-dominated points, *lower bound sets* and *upper bound sets*, formally defined by Ehrgott and Gandibleux (2007), are used instead. Given two sets $\mathcal{S}^1, \mathcal{S}^2 \subseteq \mathbb{R}^p$, we define $\mathcal{S}^1 + \mathcal{S}^2$ as the Minkowski sum, i.e. $\mathcal{S}^1 + \mathcal{S}^2 := \{s^1 + s^2 : s^1 \in \mathcal{S}^1, s^2 \in \mathcal{S}^2\}$. A set $\mathcal{S}$ is $\mathbb{R}^p_{\geqq}$-closed if the set $\mathcal{S} + \mathbb{R}^p_{\geqq}$ is closed, and $\mathbb{R}^p_{\geqq}$-bounded if there exists $s \in \mathbb{R}^p$ such that $\mathcal{S} \subset \{s\} + \mathbb{R}^p_{\geqq}$. Finally, the closure of $\mathcal{S}$ is denoted by $\text{cl}(\mathcal{S})$, and we define $\mathcal{S}_N$ as the non-dominated points of $\mathcal{S}$, i.e. $\mathcal{S}_N = \{s \in \mathcal{S} : \nexists s' \in \mathcal{S}, s' \leqq s\}$.

Definition 1.2. Given a set $\mathcal{S} \subset \mathbb{R}^p$ and its set of non-dominated points $\mathcal{S}_N$:

- a lower bound set $\mathcal{L}$ on $\mathcal{S}_N$ is an $\mathbb{R}^p_{\geqq}$-closed and $\mathbb{R}^p_{\geqq}$-bounded set such that $\mathcal{L} = \mathcal{L}_N$ and $\mathcal{S}_N \subset \mathcal{L} + \mathbb{R}^p_{\geqq}$
- an upper bound set $\mathcal{U}$ on $\mathcal{S}_N$ is an $\mathbb{R}^p_{\geqq}$-closed and $\mathbb{R}^p_{\geqq}$-bounded set such that $\mathcal{U} = \mathcal{U}_N$ and $\mathcal{S}_N \subset cl[\mathbb{R}^p \backslash (\mathcal{U} + \mathbb{R}^p_{\geqq})]$

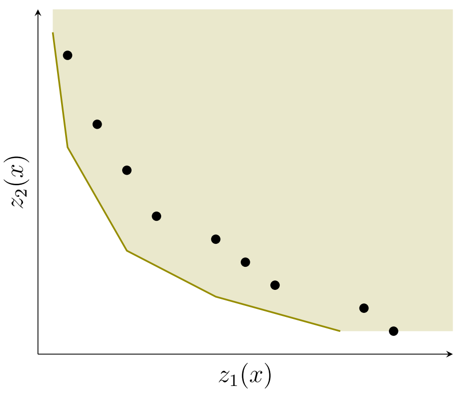
A classical lower bound for a problem with integer variables is obtained by solving the linear relaxation. Its multi-objective equivalent yields valid lower bound sets (Ehrgott and Gandibleux, 2007). The linear relaxation P^{LP} of a MOILP $P : \min\{z(x) : Ax \geqq b, x \in \mathbb{Z}^n\}$ is constructed by relaxing the integrality constraints on the variables, i.e. $P^{LP} : \min\{z(x) : Ax \geqq b, x \in \mathbb{R}^n\}$. The feasible set of the linear relaxation is a polyhedron in the decision space. Since linear functions only are considered, its image in the objective space is also a polyhedron. Hence, the lower bound set obtained by solving the multi-objective linear relaxation corresponds to the non-dominated part of a polyhedron. An example is given for $p = 2$ in Figure 1.2a. The non-dominated points of the original problem are represented by the black dots, and the lower bound set obtained by solving the linear relaxation is depicted by the thick green lines. One can observe that in particular, the non-dominated points are located in the region of the objective space that is dominated the lower bound set, i.e. in the green region on the figure.

Stronger lower bound sets can be obtained by solving the convex relaxation, i.e. by generating the convex hull of the non-dominated points. An example is given in Figure 1.2b, where the convex relaxation is shown by the thick green line, and the non-dominated points by the black dots. Once again, in accordance with Definition 1.2, the non-dominated points are located in the region that is dominated by the lower bound set, depicted by the green region. Moreover, one can observe that the supported points are part of the lower bound sets, and that the extreme points of the convex relaxation correspond to extreme supported points.

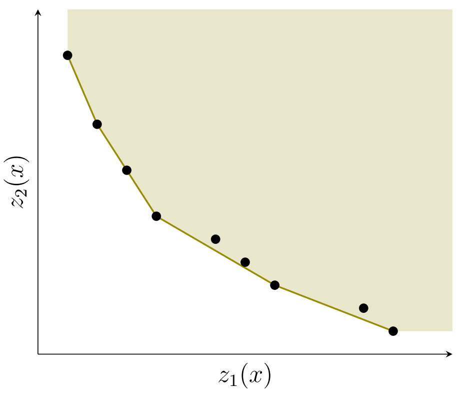
Finally, two particular lower bound sets are the ideal point and the ideal point of the linear relaxation (Ehrgott and Gandibleux, 2007).

In the single-objective case, a typical upper bound is the incumbent solution, i.e. the best known feasible solution for the problem at hand. Its multi-objective counterpart is the incumbent set, i.e. the set of known feasible solutions that are *pairwise non-dominated*. This means that there is no solution in the incumbent set that is dominated by another known feasible solution. In other words, solutions of poor quality with respect to all objectives are discarded. In Figure 1.3, an example of incumbent set for a MOILP is given. Each red circle corresponds to a known feasible solution and is part of the upper bound set. The non-dominated points are represented by the black dots, and are located in the red region, which denotes the region of the objective space that is not dominated by any point of the upper bound set.

An alternative representation of the upper bound set $\mathcal{U}$ is often used. It is called the *set of local upper bounds* (or *set of local nadir points*), and is denoted by $\mathcal{N}(\mathcal{U})$. The concept has been used in the literature for many years and was formally defined for any number of dimensions by Klamroth, Lacour, and Vanderpooten (2015). The definitions from this paper



(a) The lower bound set, depicted by the thick green line, is obtained by solving the linear relaxation. One can observe that the non-dominated points of the initial problem, represented here by the black dots, are located in the region of the objective space that is dominated by the lower bound set, i.e. in the green region.



(b) The lower bound set, depicted by the thick green line, is obtained by solving the convex relaxation. In particular, the non-dominated points of the initial problem, represented here by the black dots, are located in the region of the objective space that is dominated by the lower bound set, i.e. in the green region.

Figure 1.2: Two examples of lower bound sets.

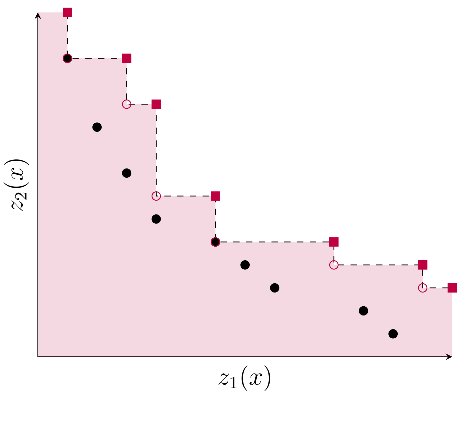


Figure 1.3: The upper bound set is depicted by the red circles, and the set of local upper bounds by the red squares. The set of non-dominated points, represented by the black dots, is located in the region of the objective space that is not dominated by any point of the upper bound set.

are used and adapted to our context here (see Definition 1.3). Let $u \in \mathbb{R}^p$, we define the *search cone* of u as $\mathcal{C}(u) = \{y \in \mathbb{R}^p : y \leqq u\}$.

Definition 1.3. Given an upper bound set $\mathcal{U}$, the corresponding set of local upper bounds $\mathcal{N}(\mathcal{U})$ is the set that satisfies:

- $cl(\mathbb{R}^p \backslash (\mathcal{U} + \mathbb{R}^p_{\geqq})) = \bigcup_{u \in \mathcal{N}(\mathcal{U})} \mathcal{C}(u)$
- $\forall u^1, u^2 \in \mathcal{N}(\mathcal{U})$, $u^1 \neq u^2$, $\mathcal{C}(u^1) \not\subset \mathcal{C}(u^2)$

In Figure 1.3, the local upper bounds are represented by the red squares. One can observe that they are located in the corner points of the red regions, and that each non-dominated point weakly dominates at least one local upper bound. Similarly, any point that does not dominate at least one local upper bound is not non-dominated and is located in the white region of the objective space.

1.1.2 Solution methods for MOILP

In this dissertation, the focus is on the resolution of MOILPs, although other problem classes such as MOMILP and MOCLP will be briefly mentioned. The algorithms to solve MOILPs can be roughly divided into two main categories: *Objective Space Search* algorithms (OSS), and *Decision Space Search* algorithms (DSS). An objective space search algorithm works by solving a series of *Single-Objective Integer Linear Programs* (SOILP) to enumerate one by one all the non-dominated points. The SOILPs are modified based on what is observed in the objective space, and their resolution is left to the powerful single-objective solvers such as CPLEX or Gurobi. On the other hand, a decision space search algorithm works by exploring and dividing the decision space to reach efficient solutions. Decision space search algorithms are typically branch-and-bound algorithms, and usually they do not rely on the use of powerful SOILP solvers. Moreover, much attention has been paid in the recent years to techniques that consider both the objective space and the decision space at the same time. This is typically done by retrieving information from the objective space to enhance a decision space search algorithm.

In Section 1.2, we will outline the basic principles behind OSS algorithms and provide a brief overview of the literature for these methods. In Section 1.3, a complete overview of branch-and-bound methods found in the literature is given.

1.2 Objective Space Search algorithms

As previously mentioned, Objective Space Search (OSS) algorithms work by solving a series of SOILP to compute, one by one, each non-dominated point. These SOILPs are often *scalarizations* of the MOILP at hand. One of the most straightforward scalarizations is the *weighted sum scalarization.* Let $P : \min\{z(x) : x \in \mathcal{X}\}$ be an MOILP, and $\lambda \in \mathbb{R}^p_{\geqq}$ a weight vector, the weighted sum scalarization of P is the single-objective problem P_λ given by $\min\{\lambda z(x) : x \in \mathcal{X}\}$.

Let x^* be an optimal solution to P_λ, and z^* its optimal value, then x^* is a weakly efficient solution of P, and if $\lambda \in \mathbb{R}^p_{>}$, x^* is efficient (Goeffrion, 1968). Besides, non-supported points cannot be obtained through the use of weighted-sum scalarizations with positive weights.

One class of OSS algorithms that relies on this scalarization technique are the so-called two-phase methods, in which all supported points are obtained solving a series of weighted sum scalarizations during the first phase. Then, all other non-dominated points, often harder to compute, are obtained using a different algorithmic approach in a second phase. In the bi-objective case, appropriate weights for the first phase can be obtained using the algorithm of Aneja and Nair (1979). The approach of Aneja and Nair (1979) and the general two-phase method was extended to the case where $p \geq 3$ by Przybylski, Gandibleux, and Ehrgott (2010). In the second phase, a large variety of algorithms can be used, such as another OSS algorithm (Clímaco and Pascoal, 2016), problem specific algorithms (Ulungu and Teghem, 1995; Przybylski et al., 2010), or even DSS algorithms (Visée, Teghem, Pirlot, and Ulungu, 1998).

Another widely used scalarization is the ϵ-constraint scalarization. Letting $\hat{k} \in \{1, ..., p\}$, the principle is to minimize $z_{\hat{k}}(x)$ under the constraints that all other objective functions $z_k(x)$, $k \in \{1, ..., p\} \backslash \{\hat{k}\}$, should be lower or equal to a certain value ϵ_k. Hence, we write that the ϵ-scalarization of P with parameters $\hat{k} \in \{1, ..., p\}$ and $\epsilon \in \mathbb{R}^p$ is the single-objective problem $P(\hat{k}, \epsilon) : \min\{z_{\hat{k}}(x) : x \in \mathcal{X}, z_k(x) \leq \epsilon_k \forall k \in \{1, ..., p\} \backslash \{\hat{k}\}\}$.

The solution of an ϵ-scalarization is a weakly efficient solution of P, and if the optimal solution of $P(\hat{k}, \epsilon)$ is unique, it is even efficient (Haimes, Lasdon, and Wismer, 1971). Moreover, each non-dominated point can be obtained by solving an ϵ-scalarization. Hence, $\mathcal{Y}_N$ can be generated using a very simple procedure that consists of solving a series of ϵ-scalarizations obtained by gradually decreasing the values in ϵ based on the solutions already obtained; and then filtering out the points that are not non-dominated.

Over the past decades, great effort has been dedicated to improving the ϵ-constraint method, e.g. by reducing or proving bounds on the number of SOILPs solved (Chalmet, Lemonidis, and Elzinga, 1986; Santis, Grani, and Palagi, 2020; Al-Rabeeah, Al-Hasani, Kumar, and Eberhard, 2020), improving the choice of the values of ϵ (Bérubé, Gendreau, and Potvin, 2009), exploring variants that guarantee the efficiency of the solutions obtained (Mavrotas, 2009; Mavrotas and Florios, 2013; Zhang and Reimann, 2014; Özlen and Azizoğlu, 2009; Kirlik and Sayın, 2014), and parallelization (Pettersson and Ozlen, 2017, 2019).

While the ϵ-constraint method works by progressively restricting the area of the objective space that is explored, other methods work by decomposing the objective space and exploring each sub-region independantly. In the bi-objective case, the various methods explore triangles or boxes in the objective space defined by adjacent non-dominated points (Lemesre, Dhaenens, and Talbi, 2007; Hamacher, Pedersen, and Ruzika, 2007; Boland, Charkhgard, and Savelsbergh, 2015; Leitner, Ljubić, Sinnl, and Werner, 2016). These areas are then subdivided into smaller pieces as long as new non-dominated points are found, or discarded if it is proven that no new non-dominated point can be found in the area. Boland, Charkhgard, and Savelsbergh (2016) and Boland, Charkhgard, and Savelsbergh (2017) developed methods specifically for the tri-objective case, namely the L-shaped method and the quadrant shrinking method. When considering three or more objective functions, Dächert and Klamroth (2015) showed that objective space decomposition methods can be extended and improved (in particular by solving fewerSOILP) by having an appropriate representation of the search region, namely using the local upper bounds. Klamroth et al. (2015) and Dächert, Klamroth, Lacour, and Vanderpooten (2017) developed algorithms designed to efficiently compute the local upper bounds, in particular in higher dimensions, thus improving the class of objective space decomposition algorithm. Tamby and Vanderpooten (2021) further improved this approach by using some properties of the ϵ-constraint method to explore more efficiently the regions defined by the local upper bounds.

Instead of exploring the objective space region by region like in objective space decomposition methods, some authors suggest to consider the entire search region at the same time, thus solving fewerSOILP. However, the search region is usually not convex (union of rectangles), and "or" constraints are used to overcome that, leading to more time consuming SOILP. This class of algorithms is often referred to as disjunctive programming (Sylva and Crema, 2004, 2008; Lokman and Köksalan, 2012; Bektaş, 2018).

Finally, other scalarizations have been used in OSS algorithms, such as the Tchebycheff scalarization. This scalarization technique consists of choosing a reference point and find the feasible point that minimizes the distance to this reference point. In principle, any norm could be used, but the infinite norm (l^{∞}-norm) is often selected, as a variant where positive weights to each objective guarantees that all non-dominated points can be reached. However, weakly non-dominated points may be generated as well with the latter approach. For more studies on the Tchebycheff scalarization, the reader is refered to Sayın and Kouvelis (2005); Ralphs, Saltzman, and Wiecek (2006); Dächert, Gorski, and Klamroth (2012); Clímaco and Pascoal (2016); Holzmann and Smith (2018); Filho, Moretti, Pato, and de Oliveira (2019).

The majority of the OSS algorithms works by solving a series of SOILP using one of the powerful commercial single-objective solvers to enumerate each non-dominated point one by one. However, the solver acts as a black-box solver, and thus potential information is lost each time it is used. For example, if the SOILPs are solved using a branch-and-bound algorithm, it is possible that a particular sub-tree is explored multiple times for different non-dominated points. This creates a lot of redundant information, since exploring the sub-tree once would be sufficient. Moreover, many of the OSS algorithms require to solve infeasible SOILP to prove that no new non-dominated point can be found under particular conditions. Depending on the problem, this can be a very time consuming operation. As an attempt to overcome these drawbacks, Decision Space Search algorithm have been developed in the literature and have received more and more attention in recent years.

1.3 Decision Space Search algorithms

The class of Decision Space Search (DSS) algorithms mostly refers to *Multi-Objective Branch-and-Bound* algorithms (MOBB). The basic principle is the same as its single-objective counterpart: a problem that is computationally difficult is placed in a root node and split into disjoint sub-problems that are stored in children nodes in the tree. One keeps sub-dividing the sub-problems until all can be solved easily, and feasible potentially non-dominated solutions are harvested and added to the upper bound set, here denoted by $\mathcal{U}$, throughout the process. The algorithm stops when all nodes have been explored, and $\mathcal{U} = \mathcal{Y}_N$.

Contrary to OSS algorithms, for which a new search tree is started from scratch at each iteration to possibly obtain a single non-dominated point, the idea behind DSS algorithms is rather to find all non-dominated points using a single search tree. In this case, branching decisions (creation of sub-problems) are made in the decision space in order to reach a complete set of efficient solutions (DSS approach), instead of solving a sequence of single objective ILPs, which are modified according to the accumulated information from previous iterations (OSS approach).

To the best of the author's knowledge, the first attempt to solve a MOILP using a single branching tree is Klein and Hannan (1982). The authors propose to optimize one of the p objectives, and add disjunctive constraints on the $p-1$ remaining ones. These constraints are created and updated based on the incumbent set at each step. In this regard, the approach is similar to the class of disjunctive programming algorithms mentioned in Section 1.2. However, the main difference is that instead of calling an external single-objective solver at each iteration, their algorithm maintains a single search tree in which they allow some of the nodes to be temporarily closed. Once a new potentially non-dominated point is obtained, some of the previously fathomed nodes may become of interest to reach the next point. Such nodes are re-opened, and the search continues from there.

In the following years, MOBB frameworks, inspired by its single-objective counterpart, started to emerge in the literature. Rapidly, a classical framework became widely used, and is still relevant nowadays (see Algorithm 1.1). Let P be a MOILP solved using a MOBB. The

Step 0: initialize the list of open nodes with the root node $\mathcal{T} \leftarrow \{\eta^0\}$; initialize the upper bound set $\mathcal{U} \leftarrow \emptyset$.
Step 1: Select a node η from $\mathcal{T}$.
Step 2: Compute a lower bound set for $P(\eta)$.
Step 3: Check whether η can be fathomed. If possible, update $\mathcal{U}$. If η is fathomed, go to Step 1.
Step 4: Split $P(\eta)$ into disjoint sub-problems. Create a new node for each and add them to $\mathcal{T}$.

Algorithm 1.1: Multi-objective B&B algorithm

search tree of a MOBB is typically represented by a list of *unexplored nodes* (or *open nodes*) $\mathcal{T}$, in which each node η corresponds to a particular sub-problem of P. The sub-problem contained in the node η is denoted by $P(\eta)$. At the beginning, $\mathcal{T}$ is initialized with the root node η^0 that contains the entire initial problem P, and with an empty upper bound set $\mathcal{U}$. Then, at each iteration, a node η is selected from $\mathcal{T}$, and a lower bound set is computed for $P(\eta)$. Afterwards, using the upper and lower bound sets at hand, a number of conditions are tested to understand whether new potentially non-dominated points can be found in $P(\eta)$. These rules are often referred to as *fathoming rules*. If it can be concluded that no new potentially non-dominated point can be found in $P(\eta)$, the node is *fathomed*, i.e. it is removed from the list of open nodes. Otherwise, $P(\eta)$ is split into disjoint sub-problems. All sub-problems are stored in child nodes of η, which are then added to the list of unexplored nodes $\mathcal{T}$, whereas η is removed from $\mathcal{T}$. The algorithm terminates when all nodes have been explored, i.e. when $\mathcal{T} = \emptyset$. The general outline of a MOBB is given in Algorithm 1.1.

Throughout the years, and particularly in the recent years, more and more research has been done to improve the different components. The remainder of this section is dedicated to present the general outline of MOBB, and to discuss the improvements proposed in the literature for each components. At the end of this section, a table gathering all the research gaps identified (see Table 1.1) is given, as well as a table showing the different characteristics of each MOBB framework identified in the literature (see Table 1.2).

1.3.1 Node selection

At each iteration, the first operation performed is the selection of an open node from $\mathcal{T}$ (Step 1). There are two classical ways to explore a tree: depth-first and breadth-first. The first is equivalent to selecting the last node added to $\mathcal{T}$, whereas the latter corresponds to exploring the oldest node of $\mathcal{T}$ first. The main advantage of these node selection rules is that they are independent of the nature of the problem, i.e. they can be used in a straightforward way in MOBB without any particular thought about its extension to the multi-objective case. For this reason, these have been widely used in the literature and are still in use at the time of writing.

In many of the first papers developing DSS algorithms, the depth-first strategy was the most used (Kiziltan and Yucaoğlu, 1983; Ramos, Alonso, Sicilia, and González, 1998; Visée et al., 1998; Mavrotas and Diakoulaki, 1998, 2005; Sourd and Spanjaard, 2008; Florios, Mavrotas, and Diakoulaki, 2010). Vincent, Seipp, Ruzika, Przybylski, and Gandibleux (2013) tested both strategies on a set of randomly generated instances, and showed that depth-first strategy performed significantly better. Parragh and Tricoire (2019), on the other hand, showed that breadth-first was more efficient on some of their problem classes. This indicates that the performance of these strategies is problem-dependant, and that the difference can be significant. Given this observation, MOBB would benefit from node selection rules that are more robust, i.e. rules where the performance relative to other rules do not drastically change when the problem class changes.

Furthermore, although the chances of exploring a non-interesting part of the search tree are lower in multi-objective optimization problems due to the multiple efficient solutions that have to be reached in the tree, there is still some room for node selection rules that could improve the performance in terms of CPU time. Stidsen, Andersen, and Dammann (2014) proposed to leave the decision to a single-objective solver. Indeed, they use lower bound sets that are obtained by solving a unique linear program, namely the linear relaxation of a weighted-sum scalarization, leading to a unique optimal value associated with each node. In that respect, their setting is similar to a single-objective branch-and-bound algorithm and thus, they can benefit from the strong rules implemented in the solver to make an appropriate decision. This strategy was reused by Stidsen and Andersen (2018) and Gadegaard, Nielsen, and Ehrgott (2019).

However, to the best of the author's knowledge, there is no comparison between depth or breadth first and other rules in the literature. This is due to the fact that alternative rules to depth and breadth first have not received a lot of attention so far. This constitutes the first research gap in Table 1.1. In particular, a promising approach would be to extend the best-bound strategy, known to be efficient for the single-objective case, to the multi-objective case. This is, however, no trivial matter due to the fact that sets are used as bounds. Indeed, it is very common to have one set that is neither better nor worse than another, but instead one is better than the other in a particular region of the objective space, and the reverse holds true in other regions. In such cases, the question of which set is the best bound is no longer clear.

1.3.2 Lower bound set computation

When a node η is selected in Step 2 of Algorithm 1.1, a lower bound set is computed for the corresponding problem $P(\eta)$. In the very first MOBB frameworks, the *minimal completion* was used as lower bound set. In the single objective case, it is computed by fixing each free variable x_i to 0 if $c_i > 0$, and to 1 otherwise. Klein and Hannan (1982) consider a single-objective version of the problem during the resolution and thus, it can be computed without further considerations required. On the other hand, Kiziltan and Yucaoğlu (1983) consider all objectives at once. Thus, the authors chose to fix x_i to 0 if $\sum\limits_{k=1}^{p} c_i^k > 0$, and to 1 otherwise instead. The resulting lower bound set is a singleton, and the corresponding solution has the advantage to be integer-valued, but it may violate the constraints of the problem.

An alternative to the minimal completion is to use the ideal point. This has the advantage of considering the constraints of the problem, leading to a more realistic estimation. Unfortunately, even the ideal point is in general not feasible. Moreover, it requires to solve p single-objective versions of the problem $P(\eta)$ (one for each objective), which can be expensive if the single-objective version is already hard to solve. Thus, the ideal point of the linear relaxation is often used instead, as for instance in Mavrotas and Diakoulaki (1998), Mavrotas and Diakoulaki (2005) and Florios et al. (2010). It requires to solve only p continuous linear programs (one for each objective), and Vincent et al. (2013) showed that it performs better than the ideal point with respect to CPU time if the problem is hard. The ideal point can still be used efficiently if the single-objective version of the problem is easy, as for example in Ramos et al. (1998), where a DSS algorithm is applied to the bi-objective minimal cost spanning tree problem. Indeed, its single-objective version can be solved in polynomial time and consequently, the ideal point can be obtained in polynomial time too.

The use of more complex lower bound sets has been popularized by Sourd and Spanjaard (2008). In their paper, the authors use the convex relaxation, and apply their framework to the bi-objective minimal cost spanning tree problem, resulting in a very promising performance. In the bi-objective case, the convex relaxation can be computed by using the algorithm of Aneja and Nair (1979), which consists in solving a series of weighted-sum scalarization with appropriate weights until all supported points have been enumerated. Przybylski et al. (2010) developed a method for the case where $p \geq 3$.

However, the convex relaxation presents two drawbacks. First, it involves the resolution of multiple (and potentially many) SOILP. As an attempt to overcome that difficulty, Sourd and Spanjaard (2008) proposed a procedure to warmstart its computation using the lower bound set from the father node. Indeed, there is no need to recompute a supported point obtained in a father node that is still feasible in the child node. However, despite this improvement, Vincent et al. (2013) showed in their experiments that for hard problems, using the convex relaxation did not necessarily lead to the best CPU times, even though it did result in the smallest search tree. Second, the main algorithms used to enumerate supported points work by inner approximations, meaning that a valid lower bound set is obtained only if the algorithm terminates. This can be problematic in nodes where the convex relaxation happens to be particularly expensive to compute, and not desirable in cases where computing only a subset of the lower bound set is sufficient to fathom a node.

A weaker but less expensive alternative to the convex relaxation is the linear relaxation. It has been widely used for MOBB tackling MOMILP with two objectives, as it naturally takes into account the continuous characteristics of $\mathcal{Y}_N$, occurring due to the non-integer variables (Vincent et al., 2013; Belotti, Soylu, and Wiecek, 2016; Adelgren and Gupte, 2022). Note that Mavrotas and Diakoulaki (1998) and Mavrotas and Diakoulaki (2005) also used the extreme points of the linear relaxation in addition to the ideal point of the linear relaxation to perform some operations in the fathoming rules. However, MOMILP is not the only domain of application for the linear relaxation. Gadegaard et al. (2019) and Parragh and Tricoire (2019) both used it successfully to solve MOILP with two objectives, together with either less expensive or stronger lower bound set. In the bi-objective case, the parametric simplex algorithm (Ehrgott, 2005) is often used to solve the linear relaxation. Parragh and Tricoire (2019) warm-started the computation by retrieving the different optimal basis from the father node, helping to reduce the total CPU time. These recent MOBB frameworks have proven the relevance of the linear relaxation when $p = 2$. However, it appears that experiments for MOBB using the linear relaxation as lower bound set has yet to be conducted for the case where $p \geq 3$. This constitutes the second gap in Table 1.1.

In recent years, solving the linear relaxation of a weighted-sum scalarization to obtain a lower bound set has also become a popular approach. The main advantage is that it is very cheap to compute compared to more complex lower bound set, as it requires to solve only one single-objective linear program, but is generally stronger than the ideal point of the linear relaxation in the bi-objective case. This idea was first introduced by Stidsen et al. (2014) and re-used by Stidsen and Andersen (2018). In the experiments of Gadegaard et al. (2019), solving the linear relaxation in particular nodes and switching to the linear relaxation of a weighted sum scalarization the rest of the time resulted in very promising performances. Parragh and Tricoire (2019) also used the integer version of the weighted sum scalarization coupled with the linear relaxation for stronger lower bound sets, which also resulted in better CPU times for the bi-objective case. In the case where $p \geq 3$, Santis, Eichfelder, Niebling, and Rocktäschel (2020) generated hyperplanes that yield valid lower bound sets. Their approach was initially developed to tackle mixed-integer convex optimization problems, implying that it can be applied to MOMILP and MOILP as well. In this context, each hyperplane is in fact equivalent to the linear relaxation of a weigthed sum scalarization, solved with weights corresponding to the normal vector of that hyperplane.

Finally, other lower bound sets have been explored for particular problem classes. Jozefowiez, Laporte, and Semet (2012) applied the MOBB to problems with two objectives, including one easy objective. This is the case for example for min-max objectives, where the number of possible values is polynomial in the input size. Then, they suggest to compute a point by fixing the easy objective to some of its possible values, and solving the single-objective linear relaxation for the hard objective. In a sense, their approach to compute lower bound sets is similar to some OSS algorithms, except that they solve continuous programs instead of integer programs. The resulting lower bound set consists of a finite set of points. Parragh and Tricoire (2019) also explored the use of strong techniques from the single-objective case such as column generation to generate strong lower bound sets.

1.3.3 Fathoming rules

In the single-objective case, there are three cases in which a node η can be fathomed. First, if no feasible point exists in $P(\eta)$, then the node is *fathomed by infeasibility*. Second, if the computation of the lower bound results in a solution that is feasible for $P(\eta)$, the node is *fathomed by optimality*. Finally, if the lower bound computed for $P(\eta)$ is greater than or equal to the upper bound, the node is *fathomed by dominance*. Each of these cases has its equivalent in the multi-objective case, but the application of the conditions usually differs depending on the lower bound set used, since it can take many different forms (singleton, finite set of points, hyperplane, polyhedron...). For the same reason, some of these fathoming rules are sometimes not applicable to particular lower bound sets.

First, fathoming by infeasibility has the most straightforward extension to the multi-objective case, since it is a condition based on the feasible set rather than on the objective functions. Let $\mathcal{X}(\eta)$ be the feasible set of the problem $P(\eta)$ corresponding to the node η. When computing the ideal point, a weighted sum scalarization, the multi-objective linear relaxation, or the convex relaxation, feasible solutions to a relaxed problem are obtained. Let $\mathcal{X}^R(\eta)$ be the feasible set of this relaxed problem. If the relaxed problem is infeasible, that is, if $\mathcal{X}^R(\eta) = \emptyset$, then so is the original problem, i.e. $\mathcal{X}(\eta) = \emptyset$. Hence, if a program solved during the computation of the lower bound set happens to be infeasible, so is $P(\eta)$, and η is fathomed by infeasibility.

Fathoming by optimality in the multi-objective case naturally requires a stronger condition, namely if all points of the lower bound set are feasible for $P(\eta)$, the node can be fathomed. The easiest case is for lower bound sets that are singletons, such as the ideal point (the reasoning is analogous for the ideal point of the linear relaxation). In this case, if the ideal point y^I is feasible, then by definition all other feasible points of $P(\eta)$ are dominated by y^I and thus, no new non-dominated points except y^I can be found in $P(\eta)$. The generalization to lower bound sets that have more than one point follows a similar reasoning. Let $\mathcal{Y}(\eta)$ denote the feasible points of $P(\eta)$, and $\mathcal{L}(\eta)$ the lower bound set computed at node η. By Definition 1.2, we have $\mathcal{Y}(\eta) \subseteq \mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}$, meaning that all feasible points of $P(\eta)$ are weakly dominated by at least one point from $\mathcal{L}(\eta)$. In the case where all points of $\mathcal{L}(\eta)$ are feasible for $P(\eta)$, this implies that each feasible point of $P(\eta)$ that is not part of $\mathcal{L}(\eta)$ is dominated by a feasible point that is already known. Hence no new non-dominated points can be found in $P(\eta)$ and the node is fathomed by optimality.

A node is fathomed by dominance if each point of the lower bound set is dominated by at least one point of the upper bound set. Fathoming by dominance is perhaps the most challenging fathoming rule, as comparing sets of different nature is not necessarily straightforward. Indeed, while the incumbent set is always a finite set of points for MOILP, the lower bound set can take various shapes, and consequently the approach chosen for fathoming by dominance depends on the latter. We distinguish between three categories of lower bound sets here: finite sets of points, complex lower bound sets, and unique hyperplanes.

Lower bound sets with a finite set of points include in particular the ideal point and the ideal point of the linear relaxation, but are not restricted to singletons. In this case, the dominance test consists in a pairwise dominance test between points of the lower and upper bound set: each known feasible point from the upper bound is tested for dominance against each point of the lower bound set. Unless the problem is unbounded, both the lower and upper bound set are finite sets of points and thus, the dominance test terminates in a finite number of steps. This dominance test was widely used, in particular in the early years of MOBB, when more complex lower bound sets were not commonly used (Klein and Hannan, 1982; Kiziltan and Yucaoğlu, 1983; Ramos et al., 1998; Visée et al., 1998; Mavrotas and Diakoulaki, 1998, 2005; Florios et al., 2010; Jozefowiez et al., 2012).

A more general rule for fathoming by dominance was established by Sourd and Spanjaard (2008). Indeed, the authors observed that at node η, if a hypersurface $\mathcal{H}$ satisfying a number of properties, such that all feasible points of $P(\eta)$ are above $\mathcal{H}$, and such that all local upper bounds are below $\mathcal{H}$, then the node can be fathomed by dominance. Sourd and Spanjaard (2008) and Gadegaard et al. (2019) showed that the convex relaxation and the linear relaxation respectively both yield valid hypersurfaces for this dominance test. In practice, the condition can be easily verified by checking if any local upper bound is located in the polyhedron $\mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}$ by using simple geometrical rules. If there is none, the node can be fathomed by dominance. Gadegaard et al. (2019) also proposed to use what they call an implicit lower bound set. It consists of checking if a local upper bound is located in $\mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}$ by solving a single-objective linear program, instead of computing explicitly the entire linear relaxation and then apply geometrical rules. However, they showed in their experiments that the explicit approach is more efficient for the bi-objective case. Although both papers restrict their analysis to the bi-objective case, the main advantage of this approach is that it is applicable with any number of objectives. Furthermore, the arbitrage between implicit and explicit lower bound sets as defined by Gadegaard et al. (2019) for this approach for the case $p \geq 3$ has not been studied in the literature yet, and is consequently still an open question.

As an alternative to this dominance test for bi-objective MOMILP, Vincent et al. (2013) proposed to compute and discard the parts of the lower bound set that are dominated by the upper bound set. If, during the procedure, it becomes an empty set, the node can be fathomed by dominance. This method has also been used by Parragh and Tricoire (2019) for MOILP. This approach is particularly suitable for MOMILP, as it naturally computes the continuous part of the non-dominated set. However, its extension to the case where $p \geq 3$ is not a trivial task. Indeed, the most complex lower bound sets and non-dominated sets in the bi-objective case consist of points and edges, which can be easily represented and handled. In particular, these objects remain convex even if a part of it is discarded. This property does not hold true when $p \geq 3$, as lower bound sets can be constituted of objects of dimension greater than 2. Although the basic principle of the method can be applied to any dimension, this makes the approach more delicate to handle from an implementation point of view. This also opens the question of appropriate representations and possibly efficient and meaningful data structures to handle the complex structure of the non-dominated set of a MOMILP with three or more objectives. To the best of the author's knowledge, this has not been addressed yet in the literature, and this constitutes the third gap in Table 1.1.

Solving a unique weighted sum (integer or linear relaxation) yields a lower bound set that is made of a unique hyperplane and is also a valid hypersurface according to the definition of Sourd and Spanjaard (2008). Because of the simple shape of this hypersurface, the dominance test can be simplified. Let $\lambda \in \mathbb{R}^p_{\geqq}$ be the weight used for the weighted sum scalarization, and z^* be its optimal value. Stidsen et al. (2014) showed that if all local upper bounds $u \in \mathcal{N}(\mathcal{U})$ satisfy $\lambda u < z^*$, then the node can be fathomed by dominance. Hence, the authors highlighted that it is sufficient to keep track of the local upper bound with the largest weighted-sum value $u^{\max}$ and compare it to z^*. The dominance test is then reduced to a simple comparison between two values: if $\lambda u^{\max} < z^*$, the node is fathomed by dominance.

Stidsen et al. (2014) further improved their approach for the bi-objective case by dividing the objective space into multiple slices, and run the MOBB in each. The idea is that a larger number of easier problems will be handled by the MOBB. Moreover, a different value for $u^{\max}$ is computed in each slice, resulting in a faster processing overall due to the improved upper bound in some of the slices.

Finally, Belotti et al. (2016) proposed an overview of existing fathoming rules and developed stronger fathoming rules for MOMILP that rely on the resolution of single-objective linear programs.

1.3.4 Creation of sub-problems

If a node η cannot be fathomed, the problem $P(\eta)$ is divided into easier, usually disjoint, sub-problems. In the single-objective case, a classical way is to select a variable x_i that has not been fixed to a specific value through previous branching decisions (also called a *free variable*). Then, two sub-problems are created with the constraints $x_i \leq t$ and $x_i \geq t + 1$ respectively, for $t \in \mathbb{Z}$. This form of branching is referred to as *decision space branching*, as sub-problems are created by adding constraints on the bounds of the decision variables.

Fortunately, the basic principle does not involve any objective function, meaning that it can be directly applied to the multi-objective case. However, the heuristics for the choice of the variable to branch on are not necessarily easily extended to the multi-objective case and consequently differ. Moreover, it is well known that branching decisions have a very large impact on the CPU time in the single-objective case. We refer the reader to Achterberg, Koch, and Martin (2005) for an overview and a performance comparison of single-objective strategies. To the best of the author's knowledge, no extensive comparison of branching rules has been carried out for the multi-objective case in the literature, and this constitutes the fourth gap in Table 1.1. Nevertheless, several rules have been proposed over the years. Kiziltan and Yucaoğlu (1983) use the minimal completion as lower bound set, which can be easily computed. Thus, they propose to branch on the variable that yields the least infeasible minimal completion, the feasibility of a solution being measured by a particular criterion provided in the paper. Vincent et al. (2013) propose to rank the variables depending on their objective coefficients prior to resolution, and to branch on the free variable with the largest rank. Another common strategy is to use problem specific rules if the problem solved is known prior to resolution (Visée et al., 1998; Sourd and Spanjaard, 2008; Florios et al., 2010; Parragh and Tricoire, 2019). Finally, similar to their node selection rule, Stidsen et al. (2014); Stidsen and Andersen (2018); and Gadegaard et al. (2019) leave the decision of the branching variable to a powerful single-objective solver. Again, this is possible in their framework because their lower bound set is obtained by solving a scalarization of the problem, i.e. a single-objective problem.

Recently, a new form of branching has emerged in the literature, namely *objective space branching*, as opposed to *decision space branching*. The principle, first proposed by Stidsen et al. (2014), consists of creating sub-problems by adding constraints on the objective functions. Slicing is one way to perform objective space branching in the bi-objective case by adding constraints in the form $z_2(x) \geq tan(\alpha_i + \Delta)z_1(x)$ and $z_2(x) \geq tan(\alpha_i - \Delta)z_1(x)$, where $alpha$ is the angle defining slice i and Δ a constant defining the span of the slice. That way, the objective space is split in multiple slices, and each slice is treated independently.

Consequently, Stidsen and Andersen (2018) proposed to parallelize the sliced MOBB. This has resulted in promising improvements in terms of CPU time.

Another method, also proposed by Stidsen et al. (2014), and improved independently by Gadegaard et al. (2019) and Parragh and Tricoire (2019), has become popular in recent years. It is often referred to as *Pareto branching*, *extended Pareto branching*, or *objective branching*. In this approach, sub-problems are created by adding upper bounds on the objective functions. The idea is to discard from the search the regions of the objective where the lower bound set is already dominated by the upper bound set, and to focus on regions where new non-dominated points could possibly be found instead. Multiple studies showed the efficiency of this technique in the bi-objective case, e.g. Stidsen et al. (2014); Gadegaard et al. (2019); Parragh and Tricoire (2019); Adelgren and Gupte (2022).

The recent success of objective space branching in bi-objective MOBB frameworks shows the value of hybrid methods, i.e. techniques that retrieve information from the objective space to enhance a decision space search algorithm. However, these methods are applied in the bi-objective case only, and no study or extensions to problems with three or more objectives exist to this day in the literature. Hence, this opens the question of objective space branching in the case where $p \geqq 3$. This constitutes the fifth gap in Table 1.1.

1.3.5 Termination of the algorithm

The branch-and-bound algorithm terminates when all nodes have been explored, and $\mathcal{U} = \mathcal{Y}_N$ holds true. In the single objective case, it is well known that branch-and-bound algorithms are often able to find solutions of great quality (or even optimal solutions), but proving optimality is expensive. To the best of the author's knowledge, there is no study of that aspect of branch-and-bound methods for the multi-objective case in the literature.

Moreover, based on that observation, the single-objective solvers often provide a measure of the quality of the best known feasible solution. One way to do it is to compare the upper bound (incumbent solution) and the lower bound with the lowest value among the open nodes, resulting in the *optimality gap g*. Hence, a decision maker can chose to stop the branch-and-bound process if a certain optimality gap is reached, thus avoiding the potentially expensive proof of optimality that is not always necessary in real world applications.

Extending quality measurements to the multi-objective case is not a trivial matter, since comparing two sets is not an obvious task, and there are multiple ways to do so. Extensions of quality measurements based on distance measures between bound sets in the bi-objective case have been proposed by Ehrgott and Gandibleux (2007). Besides, while computing the hypervolume of the search region seems to be a natural measure, it is in practice something in most cases very difficult to compute due to the complex shapes and different natures of the bound sets used. This is in particular true for the case where $p \geq 3$. In the context of the bi-objective MOMILP, Adelgren and Gupte (2022) proposed a framework that uses the Hausdorff distance to define the gap between a lower and an upper bound set, and that actively returns gap measures during the resolution process. This distance can be seen as the longest minimal distance one has to travel between any pair of points from the upper and lower bound sets. The authors also explain that this measure is more robust than the hypervolume of the search region, since it can be regarded as a worst-case measurement.

Most of the quality measures used in the bi-objective case rely on the use of both the ideal and the nadir point to normalize the values and thus, obtain meaningful and comparable numbers across instances. This opens once again the question of the case where $p \geq 3$, that has not been studied yet for such features. In particular, the nadir point is difficult to compute and may not be known until completion of the MOBB (Ehrgott, 2005). Hence, obtaining comparable values become an additional problem to address. This constitutes the sixth gap in Table 1.1.

Number	Component	Research gap
1	Node selection	Better and more robust node selection rules have yet to be developed. This is to avoid the large differences in CPU time between the traditional depth and breadth first strategies when different problem classes are solved.
2	Lower bound set	In the recent literature, using the linear relaxation resulted in great performances for bi-objective branch-and-bound frameworks. However, extensions and experiments for $p \geq 3$ have yet to be conducted.
3	Dominance test	When $p \geq 3$, the non-dominated set becomes significantly more complex. In particular, facets and faces of the non-dominated set themselves can be non-convex, which requires extra effort both in the representation and the computation. A suitable method to represent and compute $\mathcal{Y}_N$ has yet to be found.
4	Branching	No comparison between variable selection rules exists in the literature. Moreover, almost no generic variable selection rules have been explored for frameworks using the most common lower bound sets such as the linear relaxation, the convex relaxation, or the ideal point.
5	Branching	No extension of objective space branching to the case $p \geq 3$ has been proposed yet in the literature.
6	Termination	Due to the complexity of computing the nadir point when $p \geq 3$, a gap measure that is easy to compute and comparable across instances has yet to be determined.

Table 1.1: Main research gaps related to the existing literature for MOBB

Reference	Problem class	p	Node selection[1]	LB set[2]	DS branching[3]	OS branching[4]	Particularities
Klein and Hannan (1982)	MOILP	Any	–	Min compl.	–	–	Use a single objective and disjunctive constraints
Kiziltan and Yucaoğlu (1983)	MO01ILP	Any	Depth	Min compl.	Least infeasible	–	–
Ramos et. al. (1998)	Spanning Tree	2	Depth	y^I	–	–	–
Visée et. al. (1998)	Knapsack	2	Depth	Pb spec.	Pb spec.	–	Embedded in a two-phase method
Mavrotas and Diakoulaki (1998)	MO01ILP	Any	Depth	y^I, SP	–	–	Uses y^I and the extreme points of the LP relax
Mavrotas and Diakoulaki (2005)	MO01MILP	Any	Depth	y^I, SP	–	–	–
Sourd and Spanjaard (2008)	Spanning Tree	2	Depth	Convex relax.	Pb spec.	–	Introduce hypersurfaces as lower bound set
Florios et. al. (2010)	Knapsack	Any	Depth	y^I	Pb spec.	–	–
Jozefowiez et. al. (2012)	One easy objective	2	–	SP	Most fractional	–	Problems with one min-max objective
Vincent et. al. (2013)	MO01ILP	2	Depth	LP relax.	Static ranking	–	Correction of Mavrotas amd Diakoulaki (2005)
Stidsen et. al. (2014)	MO01ILP	2	SO solver	WS LP relax	SO solver	Slicing, OB	Introduce the concept of objective space branching
Belotti et al. (2016)	MOMILP	2	–	LP relax	–	–	Develop stronger fathoming rules
Stidsen and Andersen (2018)	MO01ILP	2	SO solver	WS LP relax	SO solver	Slicing, OB	Use parralelization
Parragh and Tricoire (2019)	MO01ILP	2	breadth	LP relax, WS, CG	Pb spec., –	OB	–
Gadegaard et al. (2019)	MO01ILP	2	SO solver	LR relax, WS LP relax	SO solver	OB	–
Adelgren and Gupte (2022)	MOMILP	2	Depth	LP relax	Score	OB	Implement probing for the bi-objective case

[1] Depth: Depth-first strategy, breadth: breadth-first strategy, SO solver: single objective solver, –: not specified
[2] Lower bound set. Min compl.: minimal completion, y^I: ideal point (integer or LP relax), Pb spec.: problem specific, SP: set of points, Convex relax.: convex relaxation, LP relax: linear relaxation, WS LP relax: linear relaxation of weighted-sum scalarizations, WS: weighted-sum scalarization, CG: column generation
[3] Decision space branching (variable selection). –: unspecified or fixed order, Pb spec.: problem specific, SO solver: single-objective solver, Score: attributes a score to each variables.
[4] Objective space branching. –: none. OB: objective branching / Pareto branching / extended Pareto branching

Table 1.2: MOBB frameworks for MOLP in the literature and their characteristics

1.4 Contributions and structure of the dissertation

In this thesis, a DSS algorithm to solve MOILP with three or more objective functions is designed, and gradually improved across the different chapters.

Chapter 2 develops the first MOBB that can handle any number of objectives and that uses the linear relaxation as lower bound set. In that perspective, the research gap presented in item number two from Table 1.1 is addressed in this paper. The novelty lies in the fact that an outer approximation algorithm is used to compute the linear relaxation. Furthermore, this algorithm is modified to allow warm-starting using the lower bound set from the father node, and the experiments show that this results in smaller CPU times. Statistics about the behavior of the algorithm and the complexity of the lower bound sets computed are reported and analyzed as well. Finally, challenges arising from branching on general integer variables are presented and addressed.

Chapter 3 addresses the fifth gap described Table 1.1. In particular, it focuses on the extension of objective branching to any number of objectives. First, the reasons why a straightforward extension to $p \geq 3$ is not possible are detailed, and a set of desirable properties for the created sub-problems is identified. Then, an algorithm to perform objective branching with any number of objectives is provided, and the resulting sub-problems are shown to satisfy the set of properties established. The experiments show that when $p \geq 3$, objective branching is beneficial most of the time, but also that it is not as efficient as in the bi-objective case due to the challenges related to the greater number of objectives.

Chapter 4 improves multiple key components of the MOBB framework at once. First, an extension of probing to any number of objectives is provided, and is found to be particularly effective when coupled with objective branching. In that perspective, this chapter improves the results from Chapter 3 and thus, contributes to the fifth gap from Table 1.1. Then, node selection rules based on the popular best-bound idea from the single-objective case are proposed. The experiments show that some of the described rules are better than both depth and breadth first strategies on the set of instances studied, providing a partial answer for the first gap described in Table 1.1. Finally, other components such as cut generation, the impact of problem specific variable selection rules, and enumeration of solutions are briefly studied and open the door to future research directions.

Second Chapter

Warm-starting lower bound set computations for branch-and-bound algorithms for multi objective integer linear programs

History: This chapter has been prepared in collaboration with Sune Lauth Gadegaard and Lars Relund Nielsen. It is published in European Journal of Operational Research, Volume 302 number 3, pages 909 to 924. This work was presented at the 31st European Conference on Operational Research in 2021 in Athens, Greece.

Warm-starting lower bound set computations for branch-and-bound algorithms for multi objective integer linear programs

Nicolas Forget*, Sune Lauth Gadegaard*, Lars Relund Nielsen*

* Department of Economics and Business Economics, School of Business and Social Sciences, Aarhus University, Denmark

Abstract

In this paper we propose a generic branch-and-bound algorithm for solving multi-objective integer linear programming problems. In the recent literature, competitive frameworks has been proposed for bi-objective 0-1 problems, and many of these frameworks rely on the use of the linear relaxation to obtain lower bound sets. When increasing the number of objective functions, however, the polyhedral structure of the linear relaxation becomes more complex, and consequently requires more computational effort to obtain. In this paper we overcome this obstacle by speeding up the computations. To do so, in each branching node we use information available from its father node to warm-start a Bensons-like algorithm. We show that the proposed algorithm significantly reduces the CPU time of the framework on several different problem classes with three, four and five objective functions. Moreover, we point out difficulties that arise when non-binary integer variables are introduced in the models, and test our algorithm on problems that contain non-binary integer variables too.

Keywords: multiple objective programming; branch and bound; combinatorial optimization; linear relaxation; warm-starting.

2.1 Introduction

In many real-life problems, it is usually possible to define multiple relevant objectives to optimize simultaneously. For example, one could be interested in minimizing costs, distances, traveling time, the impact on the environment, and so forth. Sometimes, it is not enough to consider only one of these objectives to obtain a satisfactory solution to a real-life problem.

Instead, several possibly conflicting objectives should be considered simultaneously. Multi-objective optimization is the field that addresses such optimization problems and as a result, produces desirable trade-offs between the conflicting objectives.

In this paper, we consider *Multi-Objective Integer Linear Problems* (*MOILP*) with p linear objectives. It is assumed that all variables in the *decision space* are integer. A special class of MOILP consists of *Multi-objective Combinatorial Optimization Problems* (*MOCOP*) with only binary variables and well-structured constraints (Nemhauser and Wolsey, 1999).

Over the past decades, various methodologies have been proposed in the literature to solve MOILPs. These methodologies can be roughly divided into two main categories: *Objective Space Search* (*OSS*) algorithms and *Decision Space Search* (*DSS*) algorithms. The principle of an OSS algorithm is to search the objective space by solving a series of single-objective problems, obtained by *scalarizing* the objective functions (Ehrgott, 2005). Hence, the power of single-objective solvers can be used to generate the optimal set of solutions (see Section 2.2 for a formal definition). Consequently, much attention has been paid to OSS methods over the years (see e.g. Ulungu and Teghem (1995); Visée et al. (1998); Sylva and Crema (2004); Ozlen, Burton, and MacRae (2014); Kirlik and Sayın (2014); Boland et al. (2017); Boland and Savelsbergh (2016); Tamby and Vanderpooten (2021)).

In contrast, a DSS algorithm searches the decision space. To the best of our knowledge, Klein and Hannan (1982) were the first to suggest a solution approach for the MOILP using a DSS algorithm. They used a unique branching tree to solve a series of single-objective integer programs, resulting in the computation of all desirable solutions. A year later, Kiziltan and Yucaoğlu (1983) proposed another general *branch-and-bound* framework. In particular, they used minimal completion, providing a lower approximation of the ideal point as a lower bound. Both of these frameworks were designed for the multi-objective case where $p \geq 2$.

In the following years, attention was paid to problem-specific methods, for example in Ulungu and Teghem (1997) and Ramos et al. (1998). In Visée et al. (1998), the authors used a multi-objective DSS algorithm embedded in an OSS algorithm, the so-called two-phase method. The next general branch-and-bound framework was developed by Mavrotas and Diakoulaki (1998), and improved in Mavrotas and Diakoulaki (2005). Their algorithm solves MOILPs with binary variables, but can also handle continuous variables in addition to binary ones. Furthermore, whereas previous branch-and-bound frameworks rely on the use of the ideal point (or an approximation hereof) as lower bound set, the authors propose to consider both the ideal point for dominance tests, and a finite set of points, namely the extreme points of the multi-objective linear relaxation, to update the upper bound set. However, Vincent et al. (2013) showed that dominated solutions may be returned, and corrected the approach of Mavrotas and Diakoulaki (2005). The use of a finite set of points as a lower bound set was further explored for specific problems in Jozefowiez et al. (2012) for the bi-objective case and in Florios et al. (2010) for the multi-objective case.

Sourd and Spanjaard (2008) were the first to use more complex lower bound sets for the bi-objective case. They proposed to use a surface (i.e. an infinite set of points) as a lower bound set instead of a finite set of points, as was the traditional method used in the literature at that time. Due to the novel nature of their lower bound set, their approach came with a new dominance test. In their framework, the lower bound set is obtained by solving the convex relaxation, which provides the convex hull of the non-dominated points contained in a specific node. Tested on spanning tree problems, the procedure produces very good lower bound sets efficiently and results in a major speed-up, but it may be less efficient on problems having a hard single-objective version, as it needs to solve multiple single-objective integer problems at each node.

Vincent et al. (2013) showed that for bi-objective problems with computationally demanding single-objective versions, the linear relaxation is often preferable in terms of computation times, even though it leads to larger branch-and-bound trees. They also showed that the linear relaxation is preferable to the ideal point and to the ideal point of the linear relaxation. Moreover, the authors proposed an extension of their branch-and-bound framework to *Bi-Objective Mixed-Integer Problems* (BOMIP) along with an alternative dominance test. The class of BOMIP was also studied by Belotti, Soylu, and Wiecek (2013), and improved in Belotti et al. (2016) who developed stronger fathoming rules. Finally, Adelgren and Gupte (2022) provided an extensive study on BOMIP and incorporated the recent knowledge of DSS algorithms in their framework.

In recent years, more attention has been paid to hybridizing decision space search and objective space search methods for the bi-objective case. A first hybrid algorithm was developed by Stidsen et al. (2014), and later refined by Stidsen and Andersen (2018). The authors used the linear relaxation of a weighted-sum scalarization as a lower bound set, which provides a weaker but computationally less expensive surface than the linear or convex relaxation. They also developed *slicing*, with the purpose of splitting the search in the objective space into several independent cones (or slices), yielding stronger upper bound sets and at the same time enabling the possibility of parallelizing the search of each slice. Finally, the authors introduced the principle of *Pareto branching*, which consists of creating sub-problems in the objective space by deriving information from the partial dominance between the lower bound set and the upper bound set.

Gadegaard et al. (2019) introduced an improved version of Pareto branching, which they named *extended Pareto branching* in their paper, and they coupled the type of branching with the use of both the linear relaxation of weighted-sum scalarizations and multi-objective linear relaxation. In parallel, Parragh and Tricoire (2019) also developed further Pareto branching, herein denoted *objective branching*. They used it together with linear relaxation, but also with stronger lower bound sets generated using a column generation approach. In both cases, promising results were shown for the bi-objective case.

It appears that the branch-and-bound frameworks developed over the last decade are competitive when solving bi-objective problems compared to state-of-the-art-OSS algorithms. This is achieved by using more sophisticated lower bound sets, stronger fathoming rules, and injecting information derived from the objective space in the method. In this paper, we are interested in developing a branch-and-bound framework that is inspired by the recent bi-objective frameworks and apply it on problems with three objective functions or more. We focus on the use of more sophisticated lower bound sets, namely the linear relaxation, and explain how we can accelerate its computation in a multi-objective branch-and-bound setting. Furthermore, it appears that although many of the recent bi-objective frameworks can be applied to integer problems, problems with binary variables (MOCOPs) are mostly studied. We consider the general integer case and devote a section to the difficulties that may arise when the variables can take arbitrary integer values. To summarize, in this paper, we:

- develop a multi-objective branch-and-bound framework that extends the bi-objective branch-and-bound literature for both combinatorial and integer problems;
- use the linear relaxation as a lower bound set by extending the work of Gadegaard et al. (2019), and using upper bound sets from Klamroth et al. (2015);
- show how redundant half-spaces from the lower bound set can be removed efficiently;
- propose a procedure to warm-start the computation of lower bound sets;
- study how warm-starting can be beneficial for other parts of the framework;
- unveil new challenges that arise when introducing integer (non-binary) variables;
- use four different problem classes including both binary and integer variables to show that warm-starting significantly reduces the total computational time.

The remainder of this paper is organized as follows: In Section 2.2 we present the preliminaries and in Section 2.3 we present a generic branch-and-bound framework for MOILPs. Section 2.4 describes how lower bound sets can be generated using a Benson-like algorithm and how such an algorithm can be modified so warm-starting becomes possible. In Section 2.5 we conduct an extensive computational study, and finally Section 2.6 concludes the paper.

2.2 Preliminaries

In multi-objective optimization, not only one but several conflicting objectives are considered simultaneously, and hence it is most often impossible to find one solution optimizing all objectives at the same time. Therefore, it is necessary to introduce operators for the comparison of points and sets. Given $y^1, y^2 \in \mathbb{R}^p$, the point y^1 *weakly dominates* y^2 ($y^1 \leqq y^2$) if $y^1_k \leq y^2_k$, $\forall k \in \{1, ..., p\}$. Moreover, we say that y^1 *dominates* y^2 ($y^1 \leqslant y^2$) if $y^1 \leqq y^2$ and $y^1 \neq y^2$. These dominance relations can be extended to sets of points as follows: Let $\mathcal{A}, \mathcal{B} \subseteq \mathbb{R}^p$, we say that $\mathcal{A}$ *dominates* $\mathcal{B}$ if for all $b \in \mathcal{B}$, there exists $a \in \mathcal{A}$ such that $a \leqslant b$. Furthermore, a subset $\mathcal{A} \subset \mathbb{R}^p$ is said to be *stable* if for any $a, a' \in \mathcal{A}$, $a \nleqslant a'$.

Consider the *Multi-Objective Integer Linear Problem* (*MOILP*) with n variables:

$$\min\{z(x) = Cx \mid x \in \mathcal{X}\} \tag{P}$$

where $\mathcal{X} = \{x \in \mathbb{N}_0^n \mid Ax \geqq b\}$ is the *feasible set* in the *decision space*. We assume that $\mathcal{X}$ is bounded (if this is not the case, it will be detected by our algorithm). The matrix $A \in \mathbb{R}^{m \times n}$ defines the coefficients of the m constraints with right-hand side $b \in \mathbb{Z}^m$. The p linear objectives are defined using the matrix $C \in \mathbb{Z}^{p \times n}$ of *objective function coefficients*. The corresponding *set of feasible objective vectors* in the *objective space* is $\mathcal{Y} = \{z(x) \mid x \in \mathcal{X}\} := C\mathcal{X}$.

In this paper, we will focus on the computation of the *non-dominated set* of points, defined as $\mathcal{Y}_N = \{y \in \mathcal{Y} \mid \nexists y' \in \mathcal{Y}, y' \leqslant y\}$. Note that $\mathcal{Y}_N$ is discrete and bounded since $z(x)$ is linear and $\mathcal{X}$ is discrete and bounded. By extension, the non-dominated part of any set $\mathcal{S} \subseteq R^p$ will be denoted by $\mathcal{S}_N = \{s \in \mathcal{S} \mid \nexists s' \in \mathcal{S}, s' \leqslant s\}$.

2.2.1 Polyhedral theory

In this section, we recall the theory presented in Nemhauser and Wolsey (1999). Let $\mathcal{H}^+ = \{y \in \mathbb{R}^p \mid \pi y \geq \pi_0\}$ denote a *half-space* in $\mathbb{R}^p$ and let $\mathcal{H} = \{y \in \mathbb{R}^p \mid \pi y = \pi_0\}$ be the corresponding *hyperplane* with normal vector π^T. A *polyhedron* $\mathcal{P} = \{y \in \mathbb{R}^p \mid Gy \geqq e\}$ is the intersection of a finite number of half-spaces and hence a closed convex set. A polyhedron $\mathcal{P} \in \mathbb{R}^p$ is of *full dimension* if the dimension of $\mathcal{P}$ is p. A half-space is *valid* if it contains $\mathcal{P}$ and *redundant* if $\mathcal{P}$ is unchanged when removed. A bounded polyhedron is called a *polytope*.

A *face* $\mathcal{F} = \{y \in \mathcal{P} \mid y \in \mathcal{H}\}$ of $\mathcal{P}$ is the intersection of $\mathcal{P}$ and a hyperplane $\mathcal{H}$ of a valid half-space $\mathcal{H}^+$. Given that $\mathcal{P}$ is of dimension p, a *facet* is a face of dimension $p-1$. The boundary of a full dimensional polyhedron $\mathcal{P}$ can be described using a finite set of facets. Let $\mathcal{P}_H = \{\mathcal{H}^+_1, \ldots, \mathcal{H}^+_k\}$ denote the *half-space representation* of $\mathcal{P}$ (the half-spaces corresponding to the facets), then $\mathcal{P} = \cap_{\mathcal{H}^+ \in \mathcal{P}_H} \mathcal{H}^+$.

A *vertex* of $\mathcal{P}$ is a face of dimension zero. The vector $r \in \mathbb{R}^p$ is a *ray* of $\mathcal{P}$ if $x + \lambda r \in \mathcal{P}$ for all $x \in \mathcal{P}$ and $\lambda \geq 0$. A ray r of $\mathcal{P}$ is said to be *extreme* if $r = \lambda_1 r^1 + \lambda_2 r^2$ where

r^1 and r^2 are rays of $\mathcal{P}$ and $\lambda_1, \lambda_2 > 0$ implies that $r^1 = \lambda r^2$ for some $\lambda > 0$. A facet of a polyhedron $\mathcal{P}$ can be described using a finite set of vertices $\mathcal{V}_F$ and extreme rays $\mathcal{R}_F$ satisfying $\mathcal{F} = \text{conv}(\mathcal{V}_F) + \{\sum_{r\in\mathcal{R}_F} \lambda_r r, \lambda \geqq 0\}$ (convex hull of vertices and rays). Since the boundary of a polyhedron consists of a finite set of facets, a *vertex-ray representation* of polytope $\mathcal{P}$ is $\mathcal{P}_V = (\mathcal{V}_P, \mathcal{R}_P)$ satisfying $\mathcal{P} = \text{conv}(\mathcal{V}_P) + \{\sum_{r\in\mathcal{R}_P} \lambda_r r, \lambda \geqq 0\}$. In general, if we use a representation of $\mathcal{P}$ using $(\mathcal{P}_H, \mathcal{P}_V)$, the sets $\mathcal{P}_H$ and $\mathcal{P}_V$ are linked together using e.g. an adjacency list so it is known which vertices are adjacent, which vertices and rays belong to which facets, and vice versa. Note that $\mathcal{P}$ is a polytope if and only if no extreme ray exists, i.e. $\mathcal{R}_F = \emptyset$ and rays can be dropped from $\mathcal{P}_V$.

The *linear relaxation* of P can be defined as:

$$\min\{z(x) = Cx \mid x \in \mathcal{X}^{LP}\} \qquad (\text{P}^{\text{LP}})$$

where $\mathcal{X}^{LP} = \{x \in \mathbb{R}^n \mid Ax \geqq b, x \geqq 0\}$. Let $\mathcal{Y}^{LP}$ denote the corresponding feasible objective vectors and $\mathcal{Y}_N^{LP}$ the non-dominated set of P^{LP}. Note that $\mathcal{Y}^{LP}$ is a polytope (Benson, 1998), and $\mathcal{Y}_N^{LP}$ corresponds to the non-dominated part of this polytope.

Consider a set $\mathcal{S} \subset \mathbb{R}^p$ and define polyhedra $\mathbb{R}^p_{\geqq} := \{y \in \mathbb{R}^p \mid y \geqq 0\}$ and $\mathcal{S} + \mathbb{R}^p_{\geqq} := \{y \in \mathbb{R}^p \mid \exists s \in \mathcal{S}, s \leqq y\}$. For the development of the branch-and-bound algorithm, it is convenient to have a description of the polyhedron $\mathcal{P}^{LP}_{\geqq} := \mathcal{Y}_N^{LP} + \mathbb{R}^p_{\geqq}$ since $\mathcal{P}^{LP}_{\geqq}$ is a full dimension polytope with vertices contained in $\mathcal{Y}_N^{LP}$. In addition to these sets, it will be convenient to define the set $\mathbb{R}^p_{\geqslant} := \{y \in \mathbb{R}^p \mid y \geqslant 0\}$.

2.2.2 Bound sets

Given a set of points $\mathcal{S} \subseteq \mathbb{R}^p$, it is possible to define lower and upper bound sets for $\mathcal{S}_N$. For this purpose, the definition from Ehrgott and Gandibleux (2007), recalled in Definition 2.1, will be used. A subset $\mathcal{S}$ is $\mathbb{R}^p_{\geqq}$-closed if $\mathcal{S} + \mathbb{R}^p_{\geqq}$ is closed, and $\mathbb{R}^p_{\geqq}$-bounded if there exists $y \in \mathbb{R}^p$ such that $\mathcal{S} \subset \{y\} + \mathbb{R}^p_{\geqq}$.

Definition 2.1. (Ehrgott and Gandibleux, 2007) Let $\mathcal{S} \subseteq \mathbb{R}^p$ be a set.

- A lower bound set $\mathcal{L}$ for $\mathcal{S}_N$ is an $\mathbb{R}^p_{\geqq}$-closed and $\mathbb{R}^p_{\geqq}$-bounded set that satisfies $\mathcal{S}_N \subset \mathcal{L} + \mathbb{R}^p_{\geqq}$, and $\mathcal{L} = \mathcal{L}_N$.
- An upper bound set $\mathcal{U}$ for $\mathcal{S}_N$ is an $\mathbb{R}^p_{\geqq}$-closed and $\mathbb{R}^p_{\geqq}$-bounded set that satisfies $\mathcal{S}_N \subset \text{cl}[\mathbb{R}^p \backslash (\mathcal{U} + \mathbb{R}^p_{\geqq})]$ and $\mathcal{U} = \mathcal{U}_N$ ($\mathcal{U}$ is stable). Here $\text{cl}(\cdot)$ denotes the closure operator.

Ehrgott and Gandibleux (2007) showed that the singleton $\{y^I\}$, denoted the *ideal point* and defined by $y^I_k = \min_{y\in\mathcal{Y}}\{y_k\}$, is a valid lower bound set for $\mathcal{Y}_N$. The same holds for the non-dominated set of the linear relaxation P^{LP} of P. Moreover, the *anti-ideal point* $\{y^{AI}\}$,

```
1: Create the root node $\eta^0$; set $\mathcal{T} \leftarrow \{\eta^0\}$ and $\mathcal{U} \leftarrow \emptyset$
2: while $\mathcal{T} \neq \emptyset$ do
3:     Select a node $\eta$ from $\mathcal{T}$ and set $\mathcal{T} \leftarrow \mathcal{T} \setminus \{\eta\}$
4:     Find a local lower bound set to $\eta$
5:     Update the upper bound set $\mathcal{U}$
6:     if $\eta$ cannot be fathomed then
7:         Branch and split $P(\eta)$ into disjoint sub-problems $(P(\eta_1), \dots, P(\eta_k))$
8:         Create child nodes of $\eta$ and set $\mathcal{T} \leftarrow \mathcal{T} \cup \{\eta_1, \dots, \eta_k\}$
9:     end if
10: end while
11: return $\mathcal{U}$
```

Algorithm 2.1: Branch-and-bound algorithm for a MOILP.

defined as $y_k^{AI} = \max_{y \in \mathcal{Y}}\{y_k\}$, yields a valid upper bound set for $\mathcal{Y}_N$. A variant of the anti-ideal point is the *nadir point*, defined as $y_k^N = \max_{y \in \mathcal{Y}_N}\{y_k\}$. The authors also showed that, in the context of a branch-and-bound algorithm, the *incumbent set*, which is the current stable set of solutions found at any point during the algorithm, is a valid upper bound set for $\mathcal{Y}_N$.

An upper bound set $\mathcal{U}$ can alternatively be described in terms of its corresponding set of *local upper bounds* $\mathcal{N}(\mathcal{U})$ (sometimes also referred to as *local nadir points*). This concept was formally defined by Klamroth et al. (2015), and their definition is recalled in Definition 2.2. Let $\mathcal{C}(u) = u - \mathbb{R}^p_{\geqq} := \{y \in \mathbb{R}^p \mid y \leqq u\}$ be the *search cone* of $u \in \mathbb{R}^p$.

Definition 2.2. (Klamroth et al., 2015) The set of local upper bounds of $\mathcal{U}$, $\mathcal{N}(\mathcal{U})$, is a set that satisfies

- $\mathrm{cl}[\mathbb{R}^p \setminus (\mathcal{U} + \mathbb{R}^p_{\geqq})] = \bigcup\limits_{u \in \mathcal{N}(\mathcal{U})} \mathcal{C}(u)$
- $\mathcal{N}(\mathcal{U})$ is minimal, i.e. there is no $u^1, u^2 \in \mathcal{N}(\mathcal{U})$, $u^1 \neq u^2$, such that $\mathcal{C}(u^1) \subseteq \mathcal{C}(u^2)$

2.3 A branch-and-bound framework for MOILP

In this section, we describe a branch-and-bound framework for MOILPs that uses the linear relaxation to obtain lower bound sets.

A general description of a *multi-objective branch-and-bound* (*MOBB*) framework for solving problem P is given in Algorithm 2.1. The algorithm manages a branching tree, $\mathcal{T}$, where each node η contains a sub-problem of P. At each node η, the sub-problem contained in η is denoted by $P(\eta)$, and its feasible set and set of feasible objective vectors are $\mathcal{X}(\eta)$ and

$\mathcal{Y}(\eta)$ respectively. Similarly, the set of non-dominated points of $P(\eta)$ is given by $\mathcal{Y}_N(\eta)$. We define analogously $\mathcal{X}^{LP}(\eta)$, $\mathcal{Y}^{LP}(\eta)$ and $\mathcal{Y}_N^{LP}(\eta)$ for the linear relaxation $P^{LP}(\eta)$ of $P(\eta)$.

A candidate set $\mathcal{T}$ is used to store nodes that are not yet explored, and is initialized with the root node that contains the full MOILP (line 1). Moreover, a global upper bound (incumbent) set is used to maintain a stable set of feasible solutions to P. The algorithm terminates when the candidate list, $\mathcal{T}$, becomes empty; that is, when it has been proven that $\mathcal{U} = \mathcal{Y}_N$.

Implementations of a MOBB algorithm may differ in the *node selection rule* (line 3), in the way the lower bound set is calculated (line 4), and in how the upper bound set is updated (line 5). Moreover, different *fathoming rules* may be used to remove a node from the candidate set (line 6). Finally, different *variable selection rules* may be used to split a *father node* into a set of *child nodes* (lines 7-8).

As node selection rule we use the so-called *breadth first* search strategy, which follows a FIFO principle, meaning that we always chose the unprocessed node that was created first. We use the non-dominated set $\mathcal{Y}_N^{LP}(\eta)$ of the linear relaxation $P^{LP}(\eta)$ as a lower bound set in each node. We use a revisited state-of-the-art version of Benson's outer approximation algorithm using warm-starting (see Section 2.4) where the polyhedron $\mathcal{Y}_N^{LP}(\eta) + \mathbb{R}^p_{\geqq}$ is found with both a half-space and vertex and ray representation. Since an integer-feasible solution to $P^{LP}(\eta)$ is feasible for $P(\eta)$, the upper bound set can be updated using the vertex representation of the lower bound set $\mathcal{Y}_N^{LP}(\eta)$ by adding vertices corresponding to integer solutions to $\mathcal{U}$ and removing any dominated points.

Different rules can be used to prune a node as well. If $P^{LP}(\eta)$ is not feasible (i.e. $\mathcal{X}^{LP}(\eta) = \emptyset$), then $P(\eta)$ is not feasible either since $\mathcal{X}(\eta) \subseteq \mathcal{X}^{LP}(\eta) = \emptyset$, and hence the node is *fathomed by infeasibility.* In the case where $\mathcal{Y}_N^{LP}(\eta) + \mathbb{R}^p_{\geqq}$ contains a single vertex y with a feasible pre-image, the node can be *fathomed by optimality* since all points in $\mathcal{Y}(\eta)$ are weakly dominated by y. Finally, if $\mathcal{Y}_N^{LP}(\eta) + \mathbb{R}^p_{\geqq}$ is dominated by $\mathcal{U}$, the node can be *fathomed by dominance.* In practice, the latter rule is checked by applying the methodology used for the bi-objective case in Sourd and Spanjaard (2008) and in Gadegaard et al. (2019), since it extends naturally to the multi-objective case. This is recalled in Lemma 2.1.

Lemma 2.1. *Let $\mathcal{U}$ be an upper bound set for $\mathcal{Y}_N$. The node η can be fathomed by dominance if for each $u \in \mathcal{N}(\mathcal{U})$, $u \notin \mathcal{Y}_N^{LP}(\eta) + \mathbb{R}^p_{\geqq}$ holds true.*

Proof. First, for any non-dominated point $y \in \mathcal{Y}_N$ of the initial problem P, there exists at least one $u \in \mathcal{N}(\mathcal{U})$ such that $y \leqq u$. Indeed, from Definition 2.1 and Definition 2.2, we have that $\mathcal{Y}_N \subset \mathrm{cl}[\mathbb{R}^p \backslash (\mathcal{U} + \mathbb{R}^p_{\geqq})]$ and $\mathrm{cl}[\mathbb{R}^p \backslash (\mathcal{U} + \mathbb{R}^p_{\geqq})] = \bigcup_{u \in \mathcal{N}(\mathcal{U})} \mathcal{C}(u)$. Thus, $\mathcal{Y}_N \subset \bigcup_{u \in \mathcal{N}(\mathcal{U})} \mathcal{C}(u)$. This implies that for each $y \in \mathcal{Y}_N$, there exists $u \in \mathcal{N}(\mathcal{U})$ such that $y \in \mathcal{C}(u)$ and consequently, by the definition of $\mathcal{C}(u)$, there exists $u \in \mathcal{N}(\mathcal{U})$ such that $y \leqq u$. Furthermore, we know that there is no $u \in \mathcal{N}(\mathcal{U})$ such that $u \in \mathcal{Y}_N^{LP}(\eta) + \mathbb{R}^p_{\geqq}$. It is not possible that $y \in \mathcal{Y}_N^{LP}(\eta) + \mathbb{R}^p_{\geqq}$ if

$u \notin \mathcal{Y}_N^{LP}(\eta) + \mathbb{R}^p_{\geqq}$, because by construction of $\mathbb{R}^p_{\geqq}$, for any set $\mathcal{S} \subset \mathbb{R}^p$ and for any $s \in \mathcal{S} + \mathbb{R}^p_{\geqq}$, $\{s\} + \mathbb{R}^p_{\geqq} \subseteq \mathcal{S} + \mathbb{R}^p_{\geqq}$. Hence, since $y \leqq u$, we have that $u \in \{y\} + \mathbb{R}^p_{\geqq}$ and thus, $u \in \mathcal{Y}_N^{LP}(\eta) + \mathbb{R}^p_{\geqq}$, which is a contradiction. This implies that necessarily, $y \notin \mathcal{Y}_N^{LP}(\eta) + \mathbb{R}^p_{\geqq}$, and as a result, no new non-dominated point can be found in sub-problem $P(\eta)$. □

If the node cannot be fathomed, $P(\eta)$ is divided into easier sub-problems. Like in the single-objective case, two disjoint sub-problems are traditionally created by using a variable selection rule to choose a variable x_i and imposing bounds on this variable. Usually, one sub-problem will be generated with the feasible set $\{x \in \mathcal{X}(\eta) \mid x_i \leq t\}$, and the other with the feasible set $\{x \in \mathcal{X}(\eta) \mid x_i \geq t + 1\}$, where $t \in \mathbb{N}$. Choosing the variable and the bound, t, is a non-trivial task, as the performance of the MOBB highly depends on these choices. In the single-objective case, the lower bound (set) usually consists of a single solution and as a result, each variable x_i takes a single value. This can be used to make an easy choice regarding the bound imposed (e.g. $t = \lfloor x_i \rfloor$). In the multi-objective case, multiple points may exist in the lower bound set, and as a consequence, a variable may take different values for different points. A trivial choice does not exist anymore, and a rule should be applied (see Section 2.5).

2.4 Linear relaxation for MOBB

In this section, we provide a strategy for accelerating the computation of the lower bound set (line 4 in Algorithm 2.1), i.e. the linear relaxation. Our methodology relies on Benson's outer approximation algorithm (Benson, 1998) and its recent refinements (see e.g. Hamel, Löhne, and Rudloff, 2013; Csirmaz, 2015; Löhne and Weißing, 2020). For this purpose, we need a formal definition of the concept of *outer approximation*.

Definition 2.3. Let $\mathcal{P}, \mathcal{Q} \subset \mathbb{R}^p$ be two polyhedra such that $\mathcal{Q}_N \subseteq \mathcal{P}$. Then $\mathcal{P}$ is an outer approximation of $\mathcal{Q}$.

An outline of a Benson-like algorithm is given in Algorithm 2.2. The algorithm works by iteratively building tighter outer approximations of $\mathcal{P}^{LP}_{\geqq} = \mathcal{Y}^{LP} + \mathbb{R}^p_{\geqq} = \mathcal{Y}^{LP}_N + \mathbb{R}^p_{\geqq}$ and starts with an initial polyhedron that contains $\mathcal{P}^{LP}_{\geqq}$. Next, half-spaces are iteratively found whose corresponding hyperplanes define facets of $\mathcal{P}^{LP}_{\geqq}$ until all the facets have been enumerated. The algorithm provides both a vertex-ray representation and a half-space representation of $\mathcal{P}^{LP}_{\geqq}$, where a pre-image is known for each of the vertices in $\mathcal{P}^{LP}_{\geqq}$.

The initialization step (line 1 of Algorithm 2.2) consists of finding an initial polyhedron that contains $\mathcal{P}^{LP}_{\geqq}$. At each iteration of Algorithm 2.2, if there exists a vertex v in the vertex-ray representation $\mathcal{P}_V$ which is not included in $\mathcal{P}^{LP}_{\geqq}$, a cutting plane should be computed

```
1: Input: A polyhedron $\mathcal{P}$ represented using $\mathcal{P}_H$ and $\mathcal{P}_V = (\mathcal{V}_P, \mathcal{R}_P)$ such that $\mathcal{P}^{LP}_{\geqq} \subseteq \mathcal{P}$
2: while $\exists\ v \in \mathcal{V}_P$ such that $v \notin \mathcal{P}^{LP}_{\geqq}$ do
3:     Compute a cutting hyperplane $\mathcal{H}$ for $v$
4:     $(\mathcal{P}_H, \mathcal{P}_V) \leftarrow$ updateP($\mathcal{P}_H$,$\mathcal{P}_V$,$\mathcal{H}$)
5: end while
6: return $(\mathcal{P}_V, \mathcal{P}_H)$
```

Algorithm 2.2: Benson's outer approximation algorithm

in order to separate v from the polyhedron. In order to check the inclusion of vertex v on line 2, the linear program $F(v)$ is solved:

$$\begin{aligned} &\min s \\ &\text{s.t. } Ax \geqq b, && (2.1) \\ &\quad Cx - s \leqq v, && (2.2) \\ &\quad x, s \geqq 0 \end{aligned}$$

If the optimal value is 0, then $v \in \mathcal{P}^{LP}_{\geqq}$ and a pre-image of v is obtained by storing the optimal values of the x variables of $F(v)$; otherwise, v is not included in $\mathcal{P}^{LP}_{\geqq}$. Let $u \in \mathbb{R}^m$ be optimal dual values corresponding to (2.1) and $w \in \mathbb{R}^p$ dual values corresponding to (2.2). Hamel et al. (2013) showed that the hyperplane $\mathcal{H} = \{y \in \mathbb{R}^p \mid w^T y = b^T u\}$ defines a facet of $\mathcal{P}^{LP}_{\geqq}$ and that $\mathcal{H}$ separates v from $\mathcal{P}^{LP}_{\geqq}$. Hence, the hyperplane $\mathcal{H}$ on line 3 can be found using the dual values of $F(v)$. Once a cutting plane $\mathcal{H}$ is computed, the outer approximation of $\mathcal{P}^{LP}_{\geqq}$ is updated using function `updateP` on line 4 of Algorithm 2.2. The loop is repeated until no vertex v can be found, and the algorithm stops ($\mathcal{P}^{LP}_{\geqq}$ has been found, line 6).

A description of `updateP` is given in Algorithm 2.3. As input, the algorithm takes the current half-space and vertex-ray representation and the cutting hyperplane. First, the vertex-ray representation is updated by examining adjacent vertices, finding new vertices of the facet of the hyperplane $\hat{\mathcal{H}}$ and removing old vertices not part of the polyhedron (line 3). Updating the vertex-ray representation using function `updateV` is known as a sub-procedure of an *online vertex enumeration problem.* A well-known technique for solving this problem is the double description method (see e.g. Fukuda and Prodon (1996)).

Next, redundant faces are removed on lines 4-12. If redundant half-spaces are not removed, many unnecessary operations will be performed, e.g. when performing dominance tests. Moreover, having no redundant half-spaces is a necessary condition for finding adjacent vertices in the vertex enumeration algorithm used (Fukuda and Prodon (1996)). Since $\mathcal{P}^{LP}_{\geqq}$ is a full-dimension polyhedron, facets are of dimension $p - 1$, and all faces with dimensions

```
1: **Input**: $(\mathcal{P}_H, \mathcal{P}_V)$ and hyperplane $\hat{\mathcal{H}}$
2: $\mathcal{P}_H \leftarrow \mathcal{P}_H \cup \{\hat{\mathcal{H}}\}$
3: $\mathcal{P}_V \leftarrow$ updateV($\mathcal{P}_H$,$\mathcal{P}_V$,$\hat{\mathcal{H}}$)
4: for all $\mathcal{H}^+ \in \mathcal{P}_H$ (defining face $\mathcal{F}$) do
5:    if $\mathcal{F}$ have $p-1$ vertices and rays or less then
6:       $\mathcal{P}_H \leftarrow \mathcal{P}_H \setminus \{\mathcal{H}^+\}$
7:    else if $p > 3$ then
8:       if all vertices and rays of $\mathcal{F}$ lies on $\hat{\mathcal{H}}$ then
9:          $\mathcal{P}_H \leftarrow \mathcal{P}_H \setminus \{\mathcal{H}^+\}$
10:      end if
11:   end if
12: end for
13: $\mathcal{P}_V \leftarrow$ relinkV($\mathcal{P}_H$,$\mathcal{P}_V$)
14: return $(\mathcal{P}_V^t, \mathcal{P}_H^t)$
```

Algorithm 2.3: Updating the outer approximation (`updateP`)

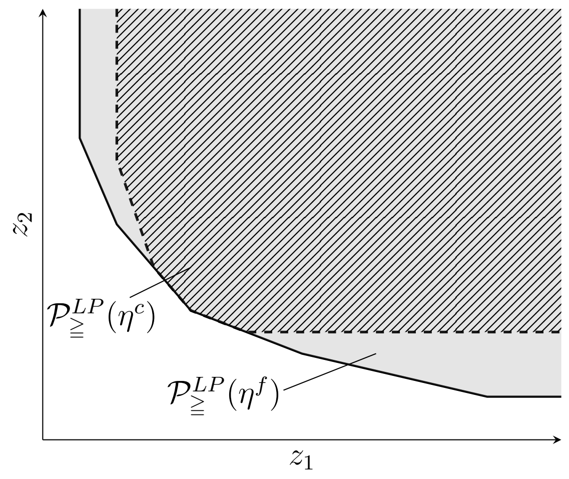
below $p-1$ are redundant. Consequently, if a facet is defined by $p-1$ vertices and rays or less, then it is redundant. This is checked on lines 5-6. Even though this is a necessary condition for any p, it is not a sufficient condition when $p > 3$. Indeed, in this case, a face of dimension 2 can be defined by more than $p-1$ vertices. A face of dimension d can be described as the intersection of at least $p-d$ hyperplanes (Nemhauser and Wolsey, 1999). Hence a face of dimension $d < p-1$ is the intersection of two or more hyperplanes. Since the input $\mathcal{P}_H$ to Algorithm 2.3 only contains facets, the only way for a facet to become a face is if it gets intersected with the new cutting hyperplane $\hat{\mathcal{H}}$ such that all of its vertices and rays are located on $\hat{\mathcal{H}}$ (lines 8-9).

Finally, since all redundant half-spaces have been removed from $\mathcal{P}_H$, we can update the adjacency list of the vertices in $\mathcal{P}_V$ using function `relinkV` on line 13. That is, using the vertex enumeration algorithm (Fukuda and Prodon (1996)).

Note that in Algorithm 2.2, only facets are generated between lines 2-5. Hence, only facets of the initial outer approximation may become redundant during the algorithm. In particular, if the initial outer approximation shares all of its facets with $\mathcal{P}_{\geqq}^{LP}$, no faces become redundant.

Lemma 2.2. *Consider Algorithm 2.2 and let $\mathcal{P}^0$ denote the initial polyhedron with half-space representation $\mathcal{P}_H^0$ (line 1). Then only half-spaces in $\mathcal{P}_H^0$ may be redundant for $\mathcal{P}_{\geqq}^{LP}$. Moreover, if $\mathcal{P}^0 = \{y_{LP}^I\} + \mathbb{R}_{\geqq}^p$ then all half-spaces in $\mathcal{P}_H^0$ are facets of $\mathcal{P}_{\geqq}^{LP}$.*

Proof. Hamel et al. (2013) showed that the cutting hyperplane $\mathcal{H}$ found on line 3 defines a facet of $\mathcal{P}_{\geqq}^{LP}$. Hence, only half-spaces in $\mathcal{P}_H^0$ may be redundant. Let $y_{LP}^I = (\hat{y}_1, \ldots, \hat{y}_p)$. If



z_2
$\mathcal{P}^{LP}_{\geqq}(\eta^c)$
$\mathcal{P}^{LP}_{\geqq}(\eta^f)$
z_1

Figure 2.1: The lower bound set of the father node η^f is an outer approximation of the lower bound set of the child node η^c.

$\mathcal{P}^0 = \{y^I_{LP}\} + \mathbb{R}^p_{\geqq}$ then the half-spaces $\{y \in \mathbb{R}^p \mid y_i \geq \hat{y}_i\}$, $i = 1, \ldots, p$ define the facets of $\mathcal{P}^0$, which are facets of $\mathcal{P}^{LP}_{\geqq}$ too. □

2.4.1 Warm-starting Benson-like algorithms in MOBB

We will now study how to improve the performance of the Benson-like algorithm embedded in a MOBB.

Lemma 2.3. *Consider a child node η^c of the father node η^f in the branch-and-bound tree of Algorithm 2.1. Then $\mathcal{P}^{LP}_{\geqq}(\eta^f) := \mathcal{Y}^{LP}_N(\eta^f) + \mathbb{R}^p_{\geqq}$ is an outer approximation of $\mathcal{P}^{LP}_{\geqq}(\eta^c) := \mathcal{Y}^{LP}_N(\eta^c) + \mathbb{R}^p_{\geqq}$.*

Proof. By construction of the problems $P(\eta^f)$ and $P(\eta^c)$, we have that $\mathcal{X}^{LP}(\eta^c) \subseteq \mathcal{X}^{LP}(\eta^f)$, which implies that $\mathcal{Y}^{LP}(\eta^c) \subseteq \mathcal{Y}^{LP}(\eta^f)$. Hence $\mathcal{P}^{LP}_{\geqq}(\eta^c) \subseteq \mathcal{P}^{LP}_{\geqq}(\eta^f)$, and since the non-dominated set of $\mathcal{P}^{LP}_{\geqq}(\eta^c)$ is $\mathcal{Y}^{LP}_N(\eta^c) \subseteq \mathcal{P}^{LP}_{\geqq}(\eta^c)$, we have that $\mathcal{P}^{LP}_{\geqq}(\eta^f)$ is an outer approximation of $\mathcal{P}^{LP}_{\geqq}(\eta^c)$. □

Due to Lemma 2.3, polyhedron $\mathcal{P}^{LP}_{\geqq}(\eta^f)$ can be used as the initial outer approximation when starting Algorithm 2.2 in a child node η^c (see Figure 2.1). That is, at any child node, it is possible to *warm-start* the computation of the linear relaxation by using the relaxation found in the father node. As a result, the total number of linear programs to be solved is expected to decrease since the only way to obtain a facet is to solve $F(v)$ for a vertex v and obtain an optimal value strictly larger than zero. Hence a facet that is present both in $\mathcal{P}^{LP}_{\geqq}(\eta^f)$ and $\mathcal{P}^{LP}_{\geqq}(\eta^c)$ will be enumerated only once, since it is already known when starting Algorithm 2.2 in child node η^c. However, some half-spaces in $\mathcal{P}_H$ may have to be removed in Algorithm 2.2 since they define non-facet faces and are therefore redundant. Due to Lemma 2.2 we have:

Corollary 2.1. *Consider Algorithm 2.1 using Algorithm 2.2 to find the lower bound set on line 4. If we use initial outer approximation $\mathcal{P} = \{y^I_{LP}(\eta^c)\} + \mathbb{R}^p_{\geqq}$ at the root node η^0, then no redundant half-spaces have to be removed from $\mathcal{P}_H$ during Algorithm 2.2. If we use initial outer approximation $\mathcal{P} = \mathcal{P}^{LP}_{\geqq}(\eta^f)$ at a child node η^c with father node η^f, then only half-spaces of $\mathcal{P}^{LP}_{\geqq}(\eta^f)$ may be redundant.*

Due to Corollary 2.1 we initialize Algorithm 2.2 with outer approximation $\{y^I_{LP}(\eta^c)\}+\mathbb{R}^p_{\geqq}$ in the root node and hence do not have to check for redundant half-spaces (lines 4-12 in Algorithm 2.3). Moreover, when using $\mathcal{P}^{LP}_{\geqq}(\eta^f)$ as initial outer approximation in a child node, only the half-spaces of $\mathcal{P}^{LP}_{\geqq}(\eta^f)$ have to be checked for redundancy.

Additional advantages of warm-starting the lower bound computations

Warm-starting Algorithm 2.2 when solving the linear relaxation in a MOBB brings additional information that can be utilized to accommodate additional speed-ups by eliminating redundant work. In the following, we present a number of observations that can help to speed-up the processing of branching nodes. In what follows, we will assume that branching is performed by adding simple bounds to variables, i.e. $x_i \leq r$ or $x_i \geq r$, for $r \in \mathbb{N}$. Further, we will refer to η^c as a child node of the father node η^f.

In Algorithm 2.2, to confirm that a vertex v is feasible, the linear program $F(v)$ has to be solved and have an optimal value of 0. However, this is not always necessary when using the polyhedron $\mathcal{P}^{LP}_{\geqq}(\eta^f)$ as a starting outer approximation when computing $\mathcal{P}^{LP}_{\geqq}(\eta^c)$:

Observation 2.1. After solving the linear relaxation in η^f, a pre-image for each vertex of $\mathcal{P}^{LP}_{\geqq}(\eta^f)$ is known. Hence, when solving the linear relaxation $P^{LP}(\eta^c)$ using $\mathcal{P}^{LP}_{\geqq}(\eta^f)$ as an initial outer approximation, it is known that $F(v) = 0$ for all $v \in \mathcal{P}^{LP}_{\geqq}(\eta^f)$ with a feasible pre-image in η^c.

This significantly reduces the number of single-objective linear programs that needs to be solved in Algorithm 2.2. In addition, when branching as stated in Section 2.3, we know that from η^f to η^c, only one constraint in the form $x_i \leq t$ or $x_i \geq t + 1$ is added, $t \in \mathbb{N}$. Hence, verifying the feasibility of a pre-image of a vertex reduces to simply comparing the value of a variable to a constant. Note that if more involved branching constraints are used, it is necessary to check that the pre-image satisfies all of them.

Observation 2.1 only holds in η^c for the vertices that belong to $\mathcal{P}^{LP}_{\geqq}(\eta^f)$. A vertex v, generated *during* the execution of Algorithm 2.2 for η^c, has to be checked for feasibility by solving $F(v)$, because we do not know a pre-image for it yet. Furthermore, we still need to solve $F(v)$ for any vertex $v \in (\mathcal{P}^{LP}_{\geqq}(\eta^f))_V$ that does not have a feasible pre-image for η^c in order to generate the cutting plane that cuts off v in line 3 of Algorithm 2.2.

Regarding the update of the upper bound set, here the incumbent set, there is also an easily achievable speed-up available: for any integer feasible vertex $v \in ((\mathcal{P}^{LP}_{\geqq}(\eta^c))_V \cap (\mathcal{P}^{LP}_{\geqq}(\eta^f))_V$ it has already been checked whether it improves the current upper bound set or not when processing η^f. Thus, only newly generated pre-images of vertices should be used when updating the upper bound set in line 5 of Algorithm 2.1 at node η^c.

Similarly, by keeping track of the cutting planes generated in node η^c, one can reduce the number of comparisons done when performing the dominance test in line 9 of Algorithm 2.1. Let $u \in \mathcal{N}(\mathcal{U})$ be a local upper bound dominated by the lower bound set in the father node η^f of η^c. This implies that $u \in \mathcal{P}^{LP}_{\geqq}(\eta^f)$, which, by using the definition of a polyhedron in terms of half-spaces, is equivalent to $u \in \bigcap_{h \in (\mathcal{P}^{LP}_{\geqq}(\eta^f))_H} h^+$. Hence, we already know that in η^c, for all the old supporting hyperplanes $h \in (\mathcal{Y}^{LP}_N(\eta^c) + \mathbb{R}^p_{\geqq})_H \cap (\mathcal{P}^{LP}_{\geqq}(\eta^f))_H$, we have $u \in h^+$. As a result, the only way for u to become non-dominated is to be located outside the half-space corresponding to one of the new facets generated in η^c. Otherwise, it remains dominated. Hence, the status of u can be determined by looking at the facets generated in η only.

To conclude, it is possible to derive additional information using Algorithm 2.2 because its starting point is exactly its ending point in its father node. Hence, we can easily keep track of how the lower bound set was modified by the new constraints generated from the father to the child node, and use this information to reduce the number of redundant operations.

2.5 Computational experiments

In this section, we present the results from our experiments conducted on MOCOs and MOILPs. The purpose of the computational study is to answer the following questions:

1. How do the different algorithm configurations perform, and which configurations perform the best?
2. How do the different variable-selection configurations perform? In particular, what is the best way to choose the bound for branching?
3. How well do the different algorithm parts perform? Especially, how much does warm-starting improve Algorithm 2.2?
4. How are leaf nodes in the branching tree fathomed?
5. How are the geometrical properties of the lower bound set evolving during the algorithm?
6. How fast can the algorithm prove optimality given a good initial upper bound set?

7. How is the performance of the MOBB algorithm compared to an objective space search algorithm?

All experiments are conducted with a time limit of one hour for solving an instance.

2.5.1 Implementation details and algorithm configurations

All algorithms are implemented in C++17 and compiled using the MSVC compiler with default options. Experiments are carried out on a personal computer with an Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz processor and 8GB of RAM memory, using Windows 10. The algorithms are available at `https://github.com/NicolasJForget/LinearRelaxationBasedMultiObjectiveBranchAndBound/tree/v1.0`. Different configurations of Algorithm 2.1 will be tested:

Node selection: A breadth-first strategy is used on line 3 of Algorithm 2.1. Preliminary experiments showed that there is no clear winner between using a depth-first or a breadth-first strategy. On the set of instances we considered, breadth-first performed slightly better on average, and we will stick to that choice for the rest of the experiments.

Calculation of lower bound set: The lower bound set on line 4 of Algorithm 2.1 is computed using two different configurations:

- `LP`: At each node η in the branching tree, $\{y^I_{LP}(\eta)\} + \mathbb{R}^p_{\geqq}$ is used as the initial outer approximation.
- `WLP`: At each node η in the branching tree with father node η^f, the lower bound set $\mathcal{P}^{LP}_{\geqq}(\eta^f)$ of the father node is used to warm-start the computation of the linear relaxation. In the root node, $\{y^I_{LP}\} + \mathbb{R}^p_{\geqq}$ is used.

Calls to the single-objective linear programming solver in Algorithm 2.2 are performed with CPLEX 12.10, using and modifying a single model for the whole branching tree. When the model is modified, the solution process is initialized using the optimal basis of the previous model solved. The lower bound polyhedron is stored using a data structure with a linked vertex-ray and a half-space representation, and updated using Algorithm 2.3.

Updating the upper bound set: The upper bound set is updated by searching for vertices in the lower bound set of a node with an integer-feasible pre-image (line 5 of Algorithm 2.1). Each time a new point is added to the upper bound set, the set of local upper bounds is also updated using the algorithm developed by Klamroth et al. (2015).

Fathoming nodes: A node is checked for fathoming (line 6 of Algorithm 2.1) by first checking if the node can be fathomed by infeasibility, then by optimality, and finally by dominance using Lemma 2.1. The dominance test terminates when one dominated local upper bound is found, or when all local upper bounds have been checked. Moreover, a non-dominated local upper bound in the father node will remain non-dominated in all of its child nodes. Hence, it is not necessary to check it again. However, this requires to keep track locally of the status of each local upper bound in $\mathcal{N}(\mathcal{U})$, which evolves globally. Preliminary experiments showed that if that resulted in minor improvements when $p = 3$ (a reduction of a few percentage points of total CPU time), the computational cost was greater than recomputing the dominance test from scratch in each node for larger p, due to the larger number of local upper bounds. Consequently, no information is kept from the father node to the children node, and the dominance test is performed from scratch at each node.

Variable selection: The variable chosen for branching on line 7 of Algorithm 2.1 is the variable that is the most often fractional among the vertex solutions in the lower bound set. If no fractional variable exists, we select the free binary variable with the average value closest to 0.5. If there is no free binary variable, a general integer variable with different maximum and minimum values is chosen. If there are ties, the variable with the lowest index is chosen. Given branching variable x_i, two child nodes are created using bound t. If x_i is binary, bound $t = 0$ is used, i.e. a rule denoted `BINARY` that branches on $x_i = t$ and $x_i = t + 1$. Given an integer branching variable x_i, a bound $t \in \mathbb{N}$ has to be chosen, and we branch using constraints $x_i \leq t$ and $x_i \geq t + 1$. Let $\{x_i^1, \ldots, x_i^k\}$ denote the sorted k values in the pre-images of all the vertices of the lower bound set. We test the configurations:

- `MED`: Choose z as the floor of the median value of $\{x_i^1, \ldots, x_i^k\}$. That is, if the values are 2.4, 3.3 and 100 then $z = \lfloor 3.3 \rfloor$, and if the values are 2.4, 3.3, 50 and 100 then $z = \lfloor 26.65 \rfloor$ (average of the two "middle" values). This is expected to result in more balanced trees, since we are more likely to discard the same number of vertices in both sub-problems.
- `MOFV`: Choose bound z such that most pre-images of vertices have $x_i \in]z, z + 1[$. The reasoning behind this rule is to discard as many vertices with a non-integer value on x_i as possible in both sub-problems created. If there is no decimal value for x_i the bound is chosen randomly in the range $[\lfloor x_i^1 \rfloor, \lceil x_i^k \rceil]$.
- `RAND`: Choose the bound randomly in the range $[\lfloor x_i^1 \rfloor, \lceil x_i^k \rceil]$.

In principle, using `LP` vs `WLP` should not affect the branching tree given all other configurations fixed. However, when the pre-image of a vertex is in fact feasible, the `LP`-configuration might produce an alternative optimum when solving $F(v)$ whereby a different pre-image of the vertex v is found. This will, potentially, lead to faster updates of the upper bound set and different search paths being followed as the pre-images are used to decide on the

Table 2.1: Instances used (480 instances in total).

Class	p[a]	n[b]	Range C[c]	$\%C$[d]	$\%A$[e]	$\#$[f]
ILP	3	10, 20, 30, 40	[-100,100]	35	80	40
ILP	4	10, 20, 30	[-100,100]	56	81	30
ILP	5	10, 20	[-100,100]	78	78	20
KP	3	10, 20, 30, 40, 50	[1,1000]	31	100	50
KP	4	10, 20, 30, 40	[1,1000]	54	100	40
KP	5	10, 20	[2,1000]	75	100	20
PPP	3	33, 39, 45, 54, 63	[1,2499]	14	3	50
PPP	4	24, 27, 33, 39, 48, 57	[1,2500]	21	4	60
PPP	5	15, 18, 24, 30, 36	[1,2500]	27	6	50
UFLP	3	42, 56, 72, 90	[1,1000]	88	3	40
UFLP	4	20, 30, 42, 56	[2,1000]	84	5	40
UFLP	5	12, 20, 30, 42	[2,1000]	81	8	40

[a] Number of objectives.
[b] Variable sizes.
[c] Range of the objective function coefficients C.
[d] Percentage of objective coefficient vectors (one vector of size p per variable) not dominated by other coefficient vectors.
[e] Percentage of non-zeros in the constraint matrix A.
[f] Number of instances.

branching variable as well as on the bound. Our test showed, however, that these differences only affect very few instances, and that the effect is negligible. As a result we only test the most promising rule (`MOFV`) for finding the bound (variable selection) for `LP`.

2.5.2 Test instances

Different problem classes are considered with 3, 4, and 5 objective functions. An overview is given in Table 2.1. For each problem class, size (number of variables), and number of objectives, 10 instances are generated. All instances can be obtained from `https://github.com/MCDMSociety/MOrepo-Forget21` and `https://github.com/MCDMSociety/MOrepo-Kirlik14`.

The problem class, denoted by ILP, consists of randomly generated MOILPs with up to 40 variables. These instances were proposed and solved in Kirlik and Sayın (2014). Note that in general, the constraint matrix for integer models modeling real-life applications is sparse and structured. We use these instances to investigate how the algorithm performs on dense and unstructured integer models. Class PPP consists of *Production Planning Problems* with up to 63 variables. Both integer and binary variables are included in the model, as well as "big M" constraints. We refer the reader to Appendix B.1 for a model description and more details regarding the ranges of the coefficients. Class KP are binary *Knapsack problems* with up to 50 variables/items. These instances were proposed and solved in Kirlik and Sayın (2014). Finally, class UFLP are *Uncapacitated Facility Location Problems* with up to 90 variables. It is a combinatorial problem (only binary variables), and the objective coefficients were generated such that the percentage of objective coefficients not dominated

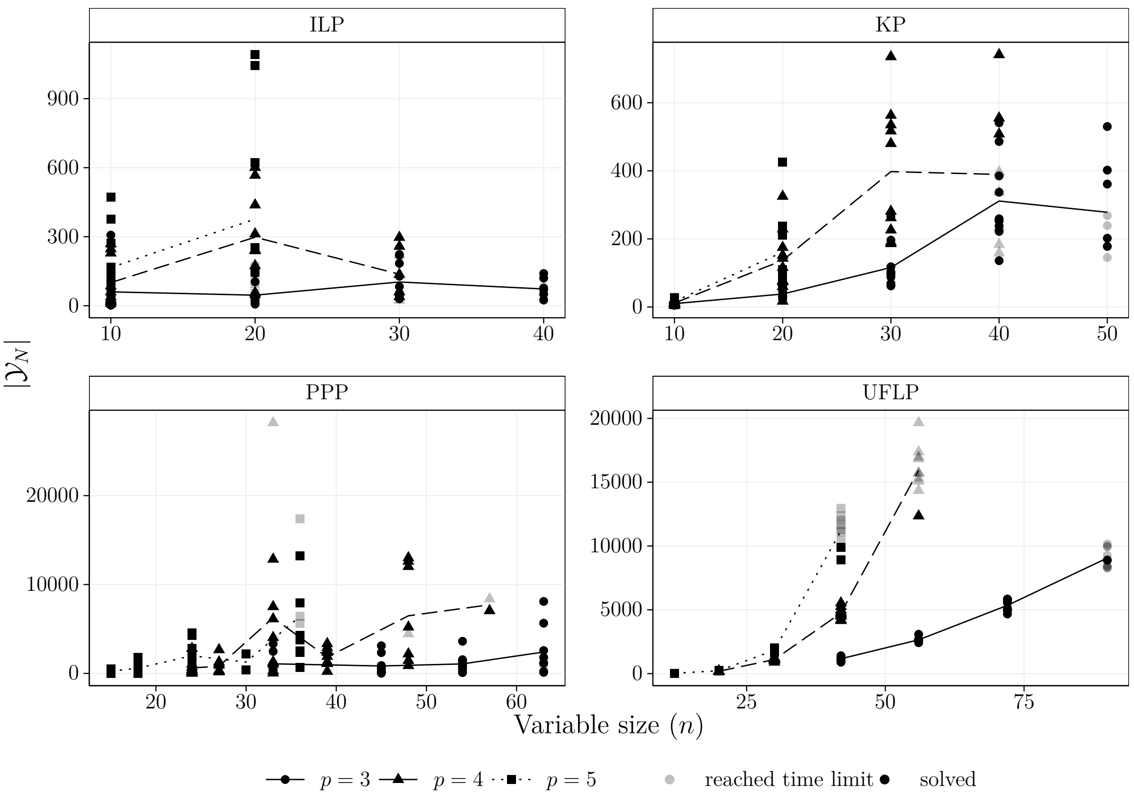


Figure 2.2: Number of non-dominated points. One point for each instance and average lines are given. Instances that have not been solved to optimality are illustrated with a transparent point.

by other coefficients is high (approx. 80%). This results in many non-dominated points. We refer the reader to Appendix B.2 for more details regarding the model and the generation of coefficients. Note that PPP and UFLP have a sparse constraint matrix compared to ILP.

In Figure 2.2 the number of non-dominated points are given for each instance. We have increased the variable size for each problem class until the size becomes so large that some instances cannot be solved within the time limit. The instances which have not been solved to optimality (19%) are illustrated with transparent points. In general the number of non-dominated points grows with variable size (n) and number of objectives (p). Note though that there may be a high variation for fixed n and p. Moreover, the variation grows with n and p. The number of non-dominated points seems to be correlated with the density of the constraint matrix (%A in Table 2.1). A dense constraint matrix may in some cases result in a small feasible solution space corresponding to few non-dominated points. However, more important factors is the range of the objective coefficients and the percentage of non-dominated objective coefficients (consider PPP and UFLP). Given a problem class, increasing these factors will result in a higher number of non-dominated points (Forget, Nielsen, and Gadegaard, 2020a). Moreover, for UFLP the number of non-dominated points grows rapidly as a function of the number of variables, which is due to the high percentage of objective coefficients not dominated by other coefficients.

2.5.3 Performance of the different algorithm configurations

First, we rank the configurations with respect to the average CPU time for all solved instances. The sequence from best to worst for the integer problems with non-binary variables (ILP and PPP) becomes `WLP-MOFV` (0%), `WLP-RAND` (8%), `LP-MOFV` (10%), and `WLP-MED` (11%), where the increase in percentages compared to the best configuration is given in parentheses. For the combinatorial problems (KP and UFLP), WLP performed on average 44% faster compared to `LP`.

A plot of the CPU time for each instance is given in Figure 2.3. Note the variation in CPU time for the 10 instances given each class and variable size. Warm-starting the computation (`WLP`) in general performs better than `LP`. On average `WLP` (using the best variable selection configuration) performed 29% faster compared to `LP`. This can also be seen in Figure 2.4 illustrating the number of solved instances given a CPU time limit. We have increased the variable size for each problem class until the size becomes so large that some instances cannot be solved within the time limit. That is, the number of instances solved before the time limit is below 100%.

In a few instances `LP` performed best (fastest in 4% of the instances). We will have a closer look at the reason for this in Section 2.5.5

2.5.4 Variable selection - Rules for choosing the bound

We are here interested in determining whether one rule for finding the bound (`MED`, `RAND`, and `MOFV`) is consistently better than the other when considering non-binary integer problems. Since the effect on the branching tree of using `LP` vs `WLP` is negligible, we will consider the `WLP` configuration here. By considering the performance profiles in Figure 2.4 we see that there is no clear winner among `MED`, `RAND`, and `MOFV`.

The `MOFV`-configuration performed best in 45% of the instances. If we compare with the second best rule for each instance, the CPU time on average increased with 11 seconds (a 3% increase).

If we take a look at the size of the branching tree in Figure 2.5, then the tree size for `MOFV` is not bigger than the one for `MED`. As a result, we use `MOFV` in the succeeding experiments, since `MOFV` is slightly faster on average and produces the smallest branching tree.

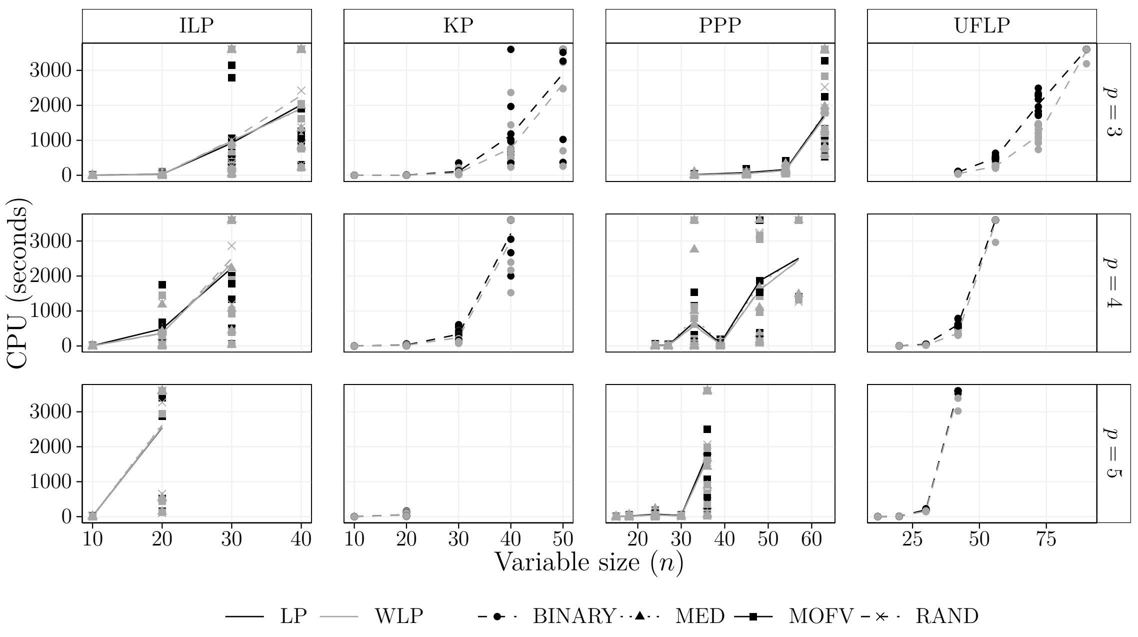


Figure 2.3: CPU times (a point for each instance) with average lines.

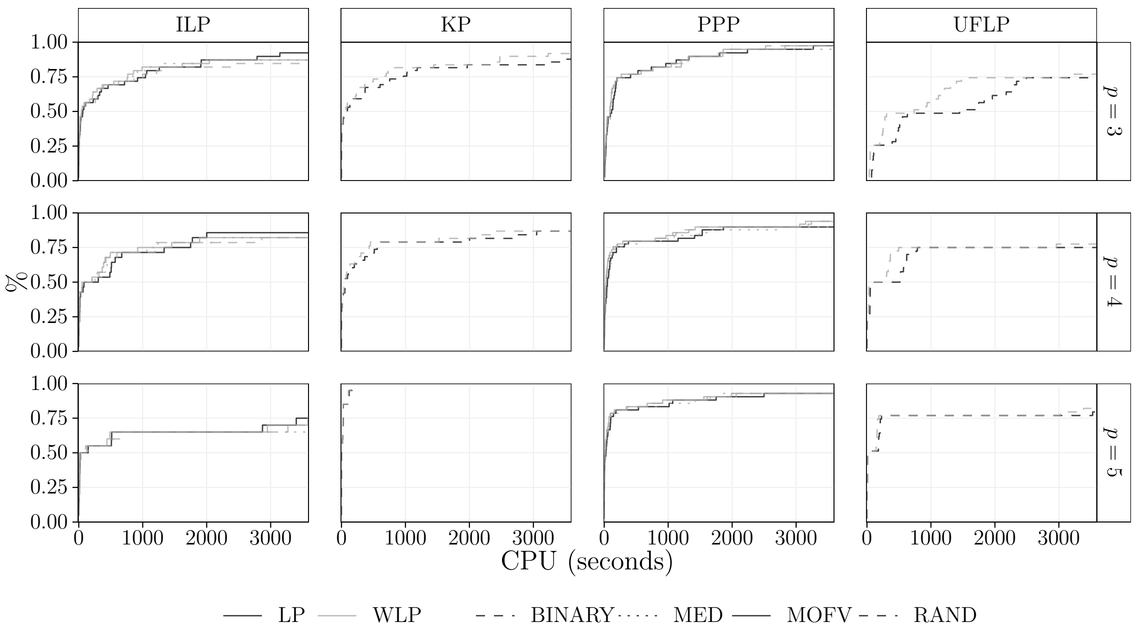


Figure 2.4: Number of instances in percent solved given cpu time (seconds). An instance is considered as unsolved if the cpu time exceed 3600 seconds (time limit). One curve is depicted for each of the configuration tested.

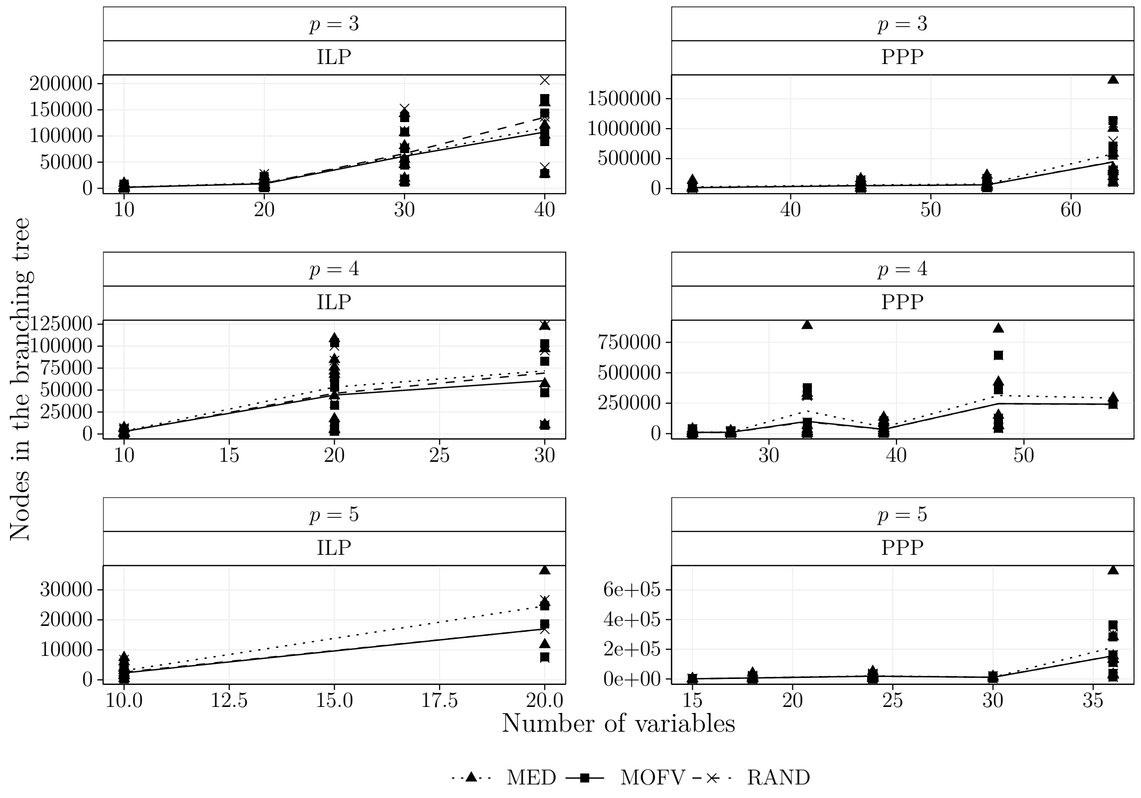


Figure 2.5: Tree size for different variable selection rules with average lines.

Table 2.2: Speed-up factor using `WLP` instead of `LP` for each problem class and number of objectives. The rule for choosing the bound is `MOFV`.

	Cpu			LPs solved		
Class	$p = 3$	$p = 4$	$p = 5$	$p = 3$	$p = 4$	$p = 5$
ILP	1.38	1.46	1.22	1.49	1.73	1.80
KP	1.43	1.36	1.12	1.51	1.55	1.50
PPP	1.37	1.69	1.74	1.55	2.15	2.61
UFLP	1.93	1.74	1.17	2.20	2.18	1.95

2.5.5 Detailed performance of different algorithm parts

In this section, we take a closer look at different parts of Algorithm 2.1. Different speed-up factors by using `WLP` instead of `LP` are given in Table 2.2. The factor is obtained by dividing the `LP` value with the `WLP` value. Only instances with both configurations solved are recorded. `WLP` are on average 1.47 times faster than `LP` with significant differences among the problem classes, e.g. for problem class UFLP, `WLP` is on average 1.61 times faster while for class ILP the speed-up is 1.35.

Most of the CPU time (95% for `LP` and 91% for `WLP`) is used on calculating the lower bound set (Algorithm 2.2) and the speed-up is mainly due to a reduction in the number of times the linear programming solver has to be called on line 2 in Algorithm 2.2. This can be seen in Table 2.2. For example, for UFLP `WLP` is 1.61 times faster and solves 2.11 times fewerlinear programs on average than `LP`. However, when using `WLP` the initial outer approximation has to be copied from the father node into the child node and managing the polyhedron is harder since we have to check for redundant half-spaces in Algorithm 2.3 (lines 4-12). As a result we have a smaller reduction in CPU times than the reduction in number of LPs solved.

Since most of the time is used for calculating the lower bound set, let us have a closer look at the relative usage of the different parts of Algorithm 2.2. An overview is given in Figure 2.6 where the different parts are:

Initialization Proportion of time used to calculate the outer approximation with polyhedron $(\{y^I_{LP}\} + \mathbb{R}^p_{\geqq}$ for `LP`, and time to copy the lower bound set from the father node for `WLP`. That is, time used to find the input on line 1 of Algorithm 2.2.

Solve LPs Proportion of time used to solve linear programs in CPLEX (line 2 of Algorithm 2.2).

Update polyhedron Proportion of time used for updating the polyhedron using Algorithm 2.3.

Other Proportion of time used on other parts of Figure 2.6, such as picking a vertex in the polyhedron (line 2), retrieving pre-images from CPLEX's ouput, checking the pre-image of a point from the father node...

First, note that the proportion of time for initialization is much higher for `LP` compared to `WLP`. For `WLP` the time for copying the lower bound set from the father is negligible. Second, note that the cost of updating the polyhedron increases with the number of objectives. This is an observation that was also made by Csirmaz (2015), who showed that in higher dimensions ($p = 10$ in their paper), updating the polyhedron was actually the bottleneck of the Benson-like algorithm. This also explains why the speed-up factors in Algorithm 2.1 in general decrease with the number of objectives while the reduction of linear programs

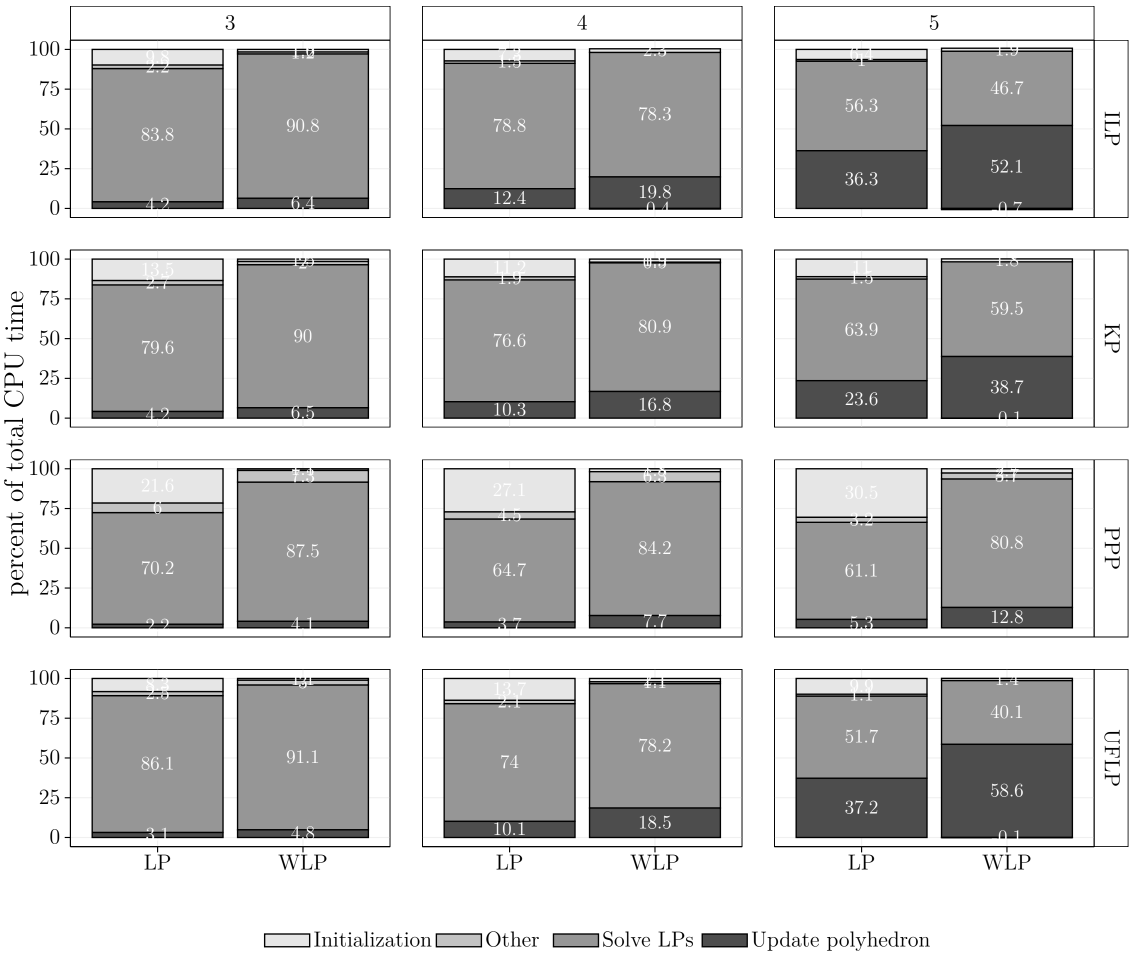


Figure 2.6: Proportion of the cpu time spent in the different components of Algorithm 2.2.

solved in general increase. Even though we solve relatively fewerlinear programs for increasing p, we have to use more time on updating the polyhedron containing the lower bound set. Hence alternative lower bound sets that does not require to manage a polyhedron, or at least fewerpolyhedral operations, may be preferred in higher dimensions. Next, note that updating the polyhedron takes a higher proportion of time for `WLP`. This is because we have to check for redundant half-spaces (lines 4-12 in Algorithm 2.3). Finally, observe that even though we have a high reduction in the number of linear programs solved for `WLP` the proportion of time used for solving linear programs is still the most predominant (except for UFLP, $p = 5$).

2.5.6 Fathoming nodes

In this section we take a closer look at how nodes are fathomed. Recall that a node is checked for fathoming by first checking if the node can be fathomed by infeasibility, next optimality, and finally by dominance. The results are illustrated in Figure 2.7 where the proportion of

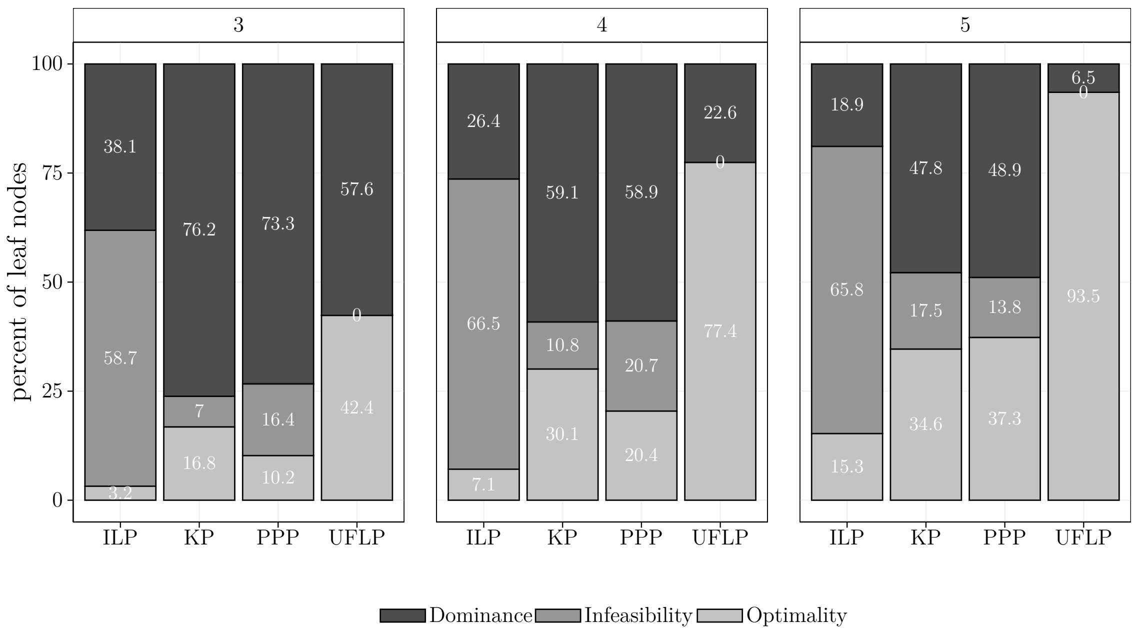


Figure 2.7: Proportion of leaf nodes fathomed by infeasibility, optimality and dominance.

leaf nodes fathomed by infeasibility, optimality and dominance are given.

It appears from Figure 2.7 that different behaviors are observed for the different problem classes. However, for all problem classes, the proportion of leaf nodes fathomed by dominance decreases as the number of objectives increases. The reason for this is that the likelihood of a point is non-dominated increases as we add dimensions to the objective space, which makes the nodes harder to prune by dominance.

Similarly, the proportion of nodes fathomed by optimality increases, which means that nodes with a unique integer vertex in the lower bound set are less rare for higher dimensional problems. Since these nodes are more likely to appear deep in the tree where many variables are fixed, it suggests that the branch-and-bound algorithm develops deeper trees when more objective functions are considered - again this is consistent with the fact that as the number of objectives increases, generating all non-dominated outcomes comes closer to a total enumeration of the decision space.

2.5.7 Geometric properties of the lower bound set during the algorithm

In this section, we take a closer look at the polyhedral properties of the lower bound sets found at each node in the branching tree during the algorithm. In Figure 2.8 statistics about the number of facets and vertices in $\mathcal{Y}_N^{LP}(\eta)$ and facets with rays in $\mathcal{P}_{\geqq}^{LP}(\eta)$, i.e. extra facets

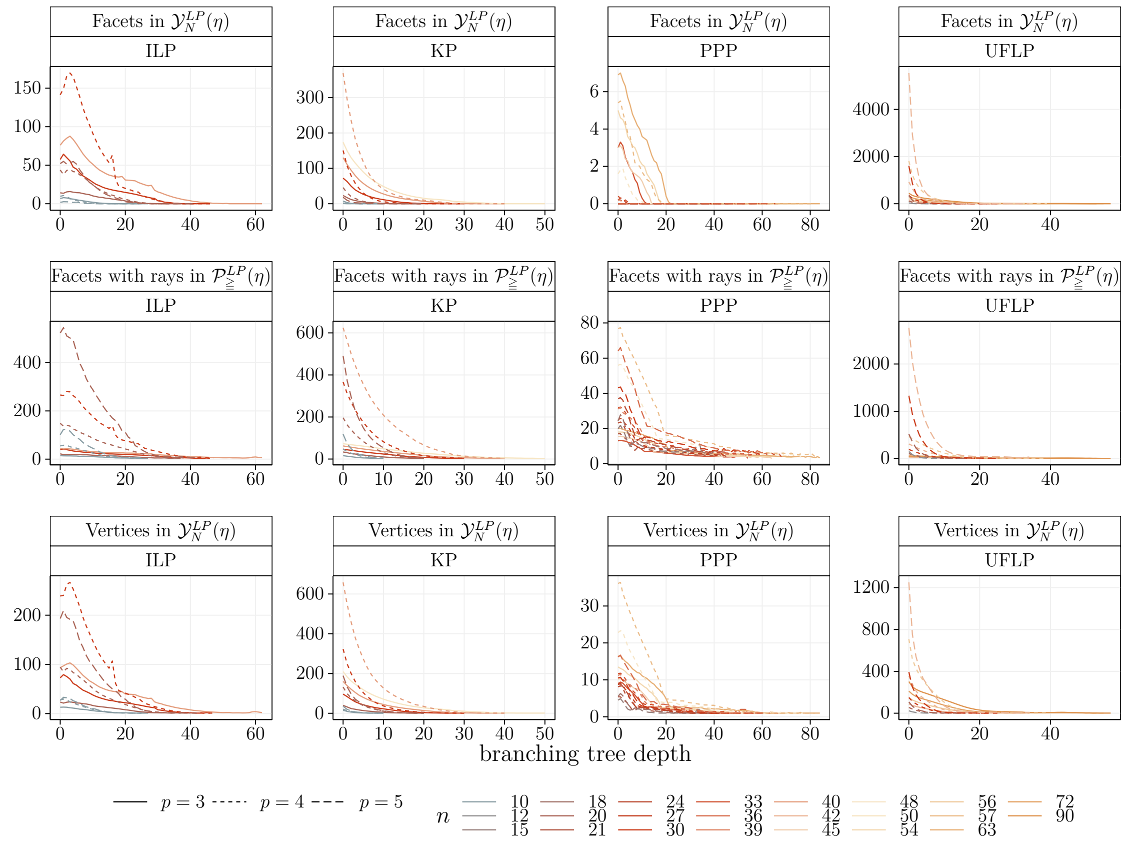


Figure 2.8: Average number of facets and vertices in the lower bound set and polyhedron calculated using Algorithm 2.2.

needed for the full-dimensional polyhedron $\mathcal{P}_{\geqq}^{LP}(\eta)$ is given. The numbers are given as a function of the depth of the branching tree with a line for each p and n.

First, note that the numbers decrease as a function of the depth of the branching tree, e.g. as we branch deeper the lower bound set has fewer facets and vertices.

Second, consider a problem class and a fixed number of variables n. As the number of objectives grows, the lower bound sets contain more facets and vertices. That is, more objectives generate more complex lower bound sets, which is mainly due to the dimension increase of the lower bound set. The same holds for fixed number of objectives p. As n grows the lower bound sets contain more facets and vertices. That is, larger problem sizes generate more complex lower bound sets.

Third, consider the decrease in the numbers as the depth grows for each problem class. For ILP and KP the numbers decrease slower compared to PPP and UFLP. This is probably due to the fact that when we branch in PPP and UFLP the subproblems become more restricted resulting in smaller lower bound sets faster (fixing a y variable in these problem classes implicitly fixes some x variables). This is not the case for ILP and KP, which do not have these implication relations between the variables and hence fixing a variable does not restrict the objective space as much.

Next, compare the number of vertices among problem classes. The number of vertices for PPP is lowest and highest for UFLP. That is, the lower bound sets in PPP are relatively simple compared to the UFLP and hence much faster to calculate and update (see Figure 2.6). Indeed, the proportion of cpu time spent in finding the initial polyhedron is very large compared to other problem classes, which suggests that only a few iterations are required to solve the linear relaxation once the initial polyhedron is found. That is particularly beneficial to `WLP` since, as we observe in Figure 2.6, warm-starting Algorithm 2.2 significantly reduces the initialization part of the algorithm.

If we consider the number of facets including rays in the polyhedron we can see that it increases rapidly with increasing p and for $p = 5$ it is higher than the number of facets (without rays) in the lower bound set (except for UFLP). That is, managing and updating the full-dimensional polyhedron instead of just the lower bound set seems to come at a higher cost as p increases.

Finally, recall the size of the non-dominated sets in Figure 2.2. Here the size of the non-dominated set for PPP is high. However, the number of vertices in the lower bound set is low. That is, problems with large non-dominated sets may generate (often weak) lower bound sets with few vertices. The opposite is also true. For ILP and KP the number of vertices is relatively higher compared to the size of the non-dominated set. That is, the relationship between the number of vertices and number of non-dominated points (i.e. the upper bound set) is problem specific.

2.5.8 Proving optimality

It is well known that it is crucial in single objective integer linear optimization to obtain a strong upper bound early during the branch-and-bound algorithm. This is to increase the potential for fathoming nodes based on the bound. Generalizing to MOILP, obtaining a strong upper bound set might also increase the potential for fathoming nodes based on dominance, thus leading to smaller trees and consequently lower CPU times.

In order to investigate the potential of such a strong upper bound set found early, we have run the algorithm where the upper bound set is initialized by $\mathcal{Y}_N$ (using `WLP` with `MOFV` or `BINARY`). That is, on line 1 of Algorithm 2.1 we replace $\mathcal{U} \leftarrow \emptyset$ with $\mathcal{U} \leftarrow \mathcal{Y}_N$. This corresponds to having a very fast and effective heuristic.

As can be seen from Figure 2.9 the reduction in CPU time is significant. On average over all the instances the speed-up factor is 2.06 meaning that on average, not providing an optimal solution at the root node, makes the cpu time increase with 106%.

As there is no significant computation time involved in generating solutions in our branch-and-bound algorithm (feasible solutions are simply harvested from integer feasible vertices

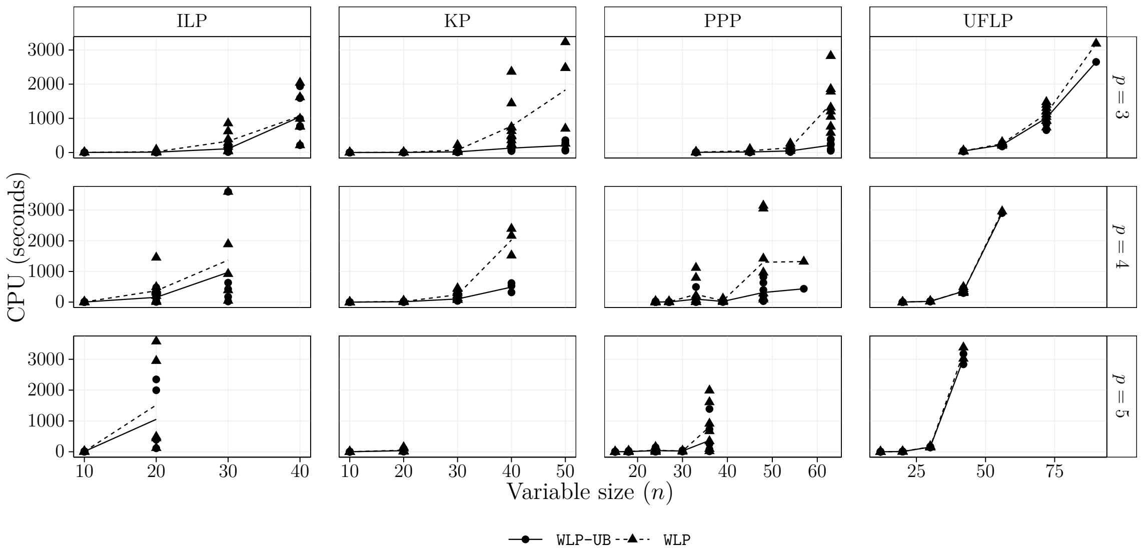


Figure 2.9: CPU times (with average lines) for `WLP` and `WLP` with the non-dominated set used as initial upper bound set in the root node (`WLP-UB`).

of the lower bound sets), the speed-up must come from the increased fathoming potential. The number of nodes in the branching tree on average decrease with 30% for `WLP-UB`. Note also, that in the `WLP-UB` configuration, the algorithm still check whether each integer feasible vertex of the lower bound sets found should enter the upper bound set, which underlines the fact that the reduction in computation time comes from the increased fathoming potential.

Concluding, it seems that, just as is the case for single objective integer programming, generating a strong upper bound set in the early stages of the algorithm may have a significant positive impact on the performance on the algorithm,

2.5.9 Performance of the MOBB algorithm compared to an objective space search algorithm

The performance of the MOBB algorithm was compared to the Objective Space Search (OSS) algorithm by Kirlik and Sayın (2014). A C++ implementation of the algorithm was obtained from Kirlik (2014). We are aware that there exist more recent OSS algorithms that outperform Kirlik (2014), in particular for $p \geq 4$ (such as Tamby and Vanderpooten (2021)); however, since no C++ implementation is available and the implementation of existing OSS algorithms is not the purpose of this paper, the algorithm by Kirlik and Sayın (2014) was used.

The current development of OSS algorithms is more mature and the algorithms may outperform MOBB algorithms. Moreover, OSS algorithms benefit from the power of single-objective MIP solvers, which have been improved over decades. Having this in mind, the

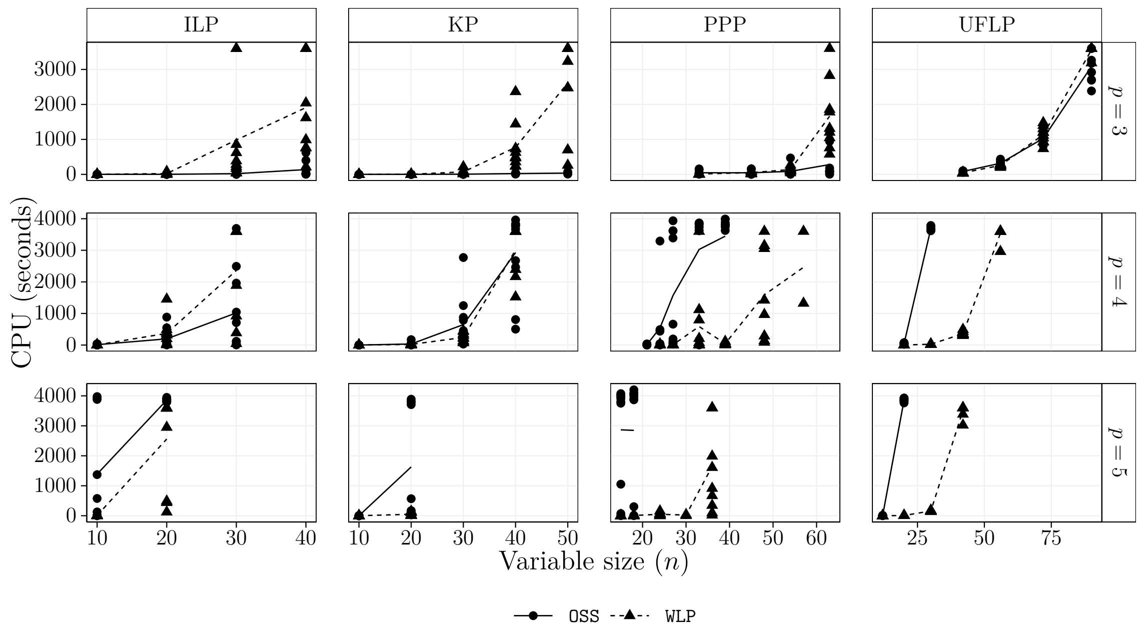


Figure 2.10: CPU times for `OSS` and `WLP` (using `MOFV`) with average lines.

purpose of this study is not necessarily to outperform the OSS algorithm, but rather to discuss and thoroughly analyse the concept of objective branching in a MOBB algorithm against an OSS algorithm. Indeed, we do believe that it will take further research before reaching a competitive MOBB algorithm, and we propose here a first step towards an efficient implementation. Moreover, our research may result in an advancement of the development of promising methods that hybridize decision space and objective space search methods.

The code by Kirlik and Sayın (2014) was updated to work with CPLEX 12.10 and compiled using the MSVC compiler with default options. The results are shown in Figure 2.10. The OSS algorithm is tested against the best configuration of our branch-and-bound framework (`WLP` with `MOFV` or `BINARY`). Different winners with respect to CPU time are observed. First, it appears that this OSS algorithm performs better on instances with three objective functions. However, as the number of objective functions increase, the trend is reversing and the MOBB algorithm becomes better than the OSS algorithm for all problem classes when $p = 5$.

When looking more closely at the case $p = 3$, note that the gap in terms of CPU time between the MOBB algorithm and the OSS algorithm is lowest for problem classes with many non-dominated points, namely PPP and UFLP. The OSS algorithm is sensitive to increasing number of non-dominated points since one single-objective ILP is solved for each non-dominated point during the OSS algorithm. Contrary, many non-dominated points may be found at the same time in some of the nodes of the branch-and-bound tree. This supports a better performance of the MOBB algorithm when handling more objectives, as more

Table 2.3: Some instances solved with a time limit increased to 4 hours. KP for $p = 5$ were not added since all instances are solved in less than one hour. ILP for $p = 5$ are not added since all instances resolved resulted in more than 4 hours CPU time for `WLP` and `OSS`.

Class	p[a]	n[b]	$\lvert\mathcal{Y}_N\rvert$[c]	`WLP`[d]	`OSS`[e]	Speedup[f]	Proportion[g]	ϵ[h]	$\bar{\epsilon}$[i]
PPP	3	63	1309	4421.91	82.63	0.02	39.7	4.3	0.5
			8093	2827.16	1213.28	0.43	100	0.0	0.0
	4	33	33453	4620.17	> 14400	> 3.12	89.6	0.4	0.0
			12844	1120.48	> 14400	> 12.50	100	0.0	0.0
	5	36	25041[j]	> 14400	> 14400	-	-	-	-
			25409	8623.28	> 14400	> 1.69	60.6	2.4	0.1
UFLP	3	90	10623	4935.04	3687.22	0.54	96.5	0.8	0.0
			8557	3661.84	2934.19	0.80	99.4	0.4	0.0
	4	56	15797	3640.48	> 14400	> 3.99	100	0.5	0.0
			20088	5748.01	> 14400	> 2.50	98.7	1.8	0.0
	5	42	12968	4258.93	> 14400	> 3.32	100	0.0	0.0
			10441	3796.83	> 14400	> 3.84	100	0.0	0.0
ILP	3	40	187	> 14400	399.88	< 0.03	3.7	11.9	4.3
			253	10176.10	127.55	0.01	7.5	10.9	2.8
	4	30	572	> 14400	772.08	< 0.06	9.3	14.7	3.7
			920	> 14400	1979.75	< 0.14	9.4	12.1	2.9
KP	3	50	557	5523.37	45.22	0.01	53.9	7.0	0.8
			383	3610.06	29.99	0.01	62.4	6.0	0.5
	4	40	901	7378.62	2672.89	0.36	10.2	10.8	2.7
			1435	5966.67	> 14400	> 2.44	22.4	11.1	2.3

[a] Number of objectives.
[b] Number of variables.
[c] Number of non-dominated points.
[d] CPU time when solving using `WLP`.
[e] CPU time when solving using `OSS`.
[f] Speedup ratio obtained by dividing the CPU time of `OSS` by the CPU time of `WLP`.
[g] Proportion of non-dominated points found after one hour of running `WLP`, in percentage.
[h] The upper bound set obtained after one hour is an ϵ-approximation of $\mathcal{Y}_N$.The number reported is 100ϵ, and can be interpreted as a percentage of the distance between y^I and y^N.
[i] The number reported is $100\bar{\epsilon}$, and can be interpreted as a percentage of the distance between y^I and y^N.
[j] As the instance was solved by none of the algorithm, the size of the upper bound set at the 4 hours time limit using `WLP` is reported instead.

non-dominated points are expected when considering more objectives.

To test the algorithms with longer running time, we extended the time limit to 4 hours, and considered two unsolved instances (either for `WLP` or `OSS`) for each problem class and number of objectives. The two instances with the lowest and highest upper bound set was picked among the unsolved instances. If only one instance was unsolved, we chose the instance solved with the largest cpu time for `WLP` for the second instance. The results are given in Table 2.3 containing CPU times, speedup ratios and approximation measures.

An *ϵ-approximation* of $\mathcal{Y}_N$ is a set $\bar{\mathcal{Y}}$ such that for each $y \in \mathcal{Y}_N$, there exists $\bar{y} \in \bar{\mathcal{Y}}$ such that $\bar{y} - \epsilon\Delta \leqq y$, where $\Delta = y^N - y^I$. This can be interpreted as the maximal proportion of the distance between the ideal and the nadir point one need to shift the points in $\bar{\mathcal{Y}}$ to satisfy this condition. To find ϵ, we extracted the the upper bound set after one hour (denoted by $\bar{\mathcal{U}}$), and calculated the ϵ needed such that $\bar{\mathcal{U}}$ is an ϵ-approximation of $\mathcal{Y}_N$. We also reported the average value $\bar{\epsilon}$ of ϵ needed for for each non-dominated point, i.e. $\bar{\epsilon} = \frac{1}{|\mathcal{Y}_N|} \sum_{y \in \mathcal{Y}_N} \epsilon_y$, where ϵ_y is the minimal value $\hat{\epsilon}$ such that there exists $\bar{y} \in \bar{\mathcal{U}}$ such that $\bar{y} - \hat{\epsilon}\Delta \leqq y$.

Consider Table 2.3. First, observe even beyond the one hour time limit, the MOBB algorithm exhibit the same behavior as when given the limit. The OSS algorithm performs better on instances with three objective functions and as the number of objective functions increase, the trend is reversing and the MOBB algorithm becomes better than the OSS algorithm. Indeed, MOBB is most competitive compared to the OSS algorithm on problems that exhibit a large number of non-dominated points.

Second, for UFLP, the proportion of non-dominated points obtained within one hour is very high even though the total computation time is not necessarily that close to one hour. This shows that for this problem class, the MOBB algorithm spend a lot of time obtaining a few missing non-dominated points and trying to prove optimality. This observation confirms the results from Figure 2.9, where having $\mathcal{Y}_N$ as upper bound set did not significantly reduce the total CPU time for UFLP. Also, it implies that in some cases, stopping the MOBB prematurely is likely to provide a good approximation of $\mathcal{Y}_N$. This is not necessarily the case if an OSS algorithm is stopped early, since at most one non-dominated point is generated at each iteration whereas multiple solutions can be harvested at a single node in a MOBB algorithm.

Next, the approximation found is of good quality for UFLP and PPP. The value of $\bar{\epsilon}$ is very close to 0, which implies that on average, $\bar{\mathcal{U}}$ is very close to $\mathcal{Y}_N$. For KP and in particular ILP the approximation is of lower quality but still relatively good.

Finally, observe that $\bar{\epsilon}$ is significantly lower than ϵ. This suggest that in some part of the objective space, the approximation is of good quality, but there are may be a few regions in which no good feasible solution is found yet. Identifying such regions and intensifying the search in those may be beneficial in case of an early stop of the MOBB algorithm.

2.6 Conclusion

In Section 2.3, we implemented a branch-and-bound algorithm that can solve any MOILP with any number of objectives. It was inspired by the recent successful bi-objective frameworks found in the literature, and then adapted to the multi-objective case. In particular, it was based on the use of linear relaxations to generate lower bound sets, and used a Benson-like algorithm to do so. We also pointed out that in case integer variables exists in the problem solved, we need to chose the bound imposed on the branching variable in the child nodes in addition the variable to branch on. This decision is not trivial anymore when there exist two points or more in the lower bound set. We tested three different rules in Section 2.5 and showed that despite the fact that they performed quite similarly in practice, there are instances where the choice has a significant impact in terms of total CPU time.

Moreover, in Section 2.4 we proposed a way to accelerate the computation of the linear relaxation in the specific context of a multi-objective branch-and-bound algorithm. It relies on the use of the lower bound set from the father node to warm-start the solution process in the current node. Our experiments showed that this led to a great reduction in the number of calls to the single-objective linear programming solver, which resulted in a significant speed-up for most of our instances. However, warm-starting the lower bound set computation comes at a greater cost of managing the polyhedra coresponding to the lower bound sets. A consequence of that is a decrease in the speed-up as the number of objective functions increases.

This latter observation suggests that for high-dimensional problems, it may be preferred to use a lower bound set that does not require as many polyhedral operations as the one used in this paper. An alternative approach could be to use an implicit lower bound set as defined in Gadegaard et al. (2019) instead of explicitly computing the linear relaxation. In such an approach, line 4 of Algorithm 2.1 is skipped, and a linear program similar to $F(u)$ is solved to check whether a local upper bound u is dominated by the lower bound set.

Besides, in the recent bi-objective literature, methods that use information from the objective space to enhance the DSS algorithm have proven to be very efficient. Extending these concepts to the case where $p \geq 3$ may then be of great interest and may potentially result in an even more efficient DSS algorithm.

Finally, similar to the single-objective case, the branch-and-bound algorithm could be stopped early to obtain an approximation of $\mathcal{Y}_N$. Our computation study showed that the upper bound set obtained after stopping the MOBB early could provide a very good approximation of $\mathcal{Y}_N$, but also that there could be some regions of the objective space in which the quality of the approximation decreases. We believe that developing a procedure that intensifies the search in such regions could be beneficial to the MOBB algorithm, in the sense that approximations of better quality could be obtained if the algorithm is stopped early. A possible approach would be for example to design an appropriate node selection rule.

2.A Notes on the the unbounded case

It was assumed in the paper that the MOILP under scrutiny had a bounded feasible set. In this appendix we will briefly demonstrate how the algorithm developed in this paper is able to detect unbounded MOILPs.

As oppose to MOCO problems, when introducing a general integer variable $x_i \in \mathbb{N}$ in the model, there is not necessarily finite bounds given on the value of x_i, and, depending on the other constraints of the problem, x_i could take an infinite number of values. In the single-objective case, an optimization problem $\min\{z(x) \mid x \in \mathcal{X}\}$ is said to be unbounded if, for all $l \in \mathbb{R}$ there exists an $x \in \mathcal{X}$ such that $z(x) < l$. In the single-objective case, detecting unbounded integer programs was already studied several decades ago. In particular, the following theorem has been stated:

Theorem 2.2 (Byrd, Goldman, and Heller (1987))**.** *Let P be a single-objective integer program with a constraint matrix denoted by A. If A has rational entries only and if P^{LP}, the linear relaxation of P, is unbounded; then P is either infeasible or unbounded .*

For the remainder of the discussion, we will only consider constraint matrices with rational coefficients. In order to investigate unboundedness in MOILPs we formally define an unbounded MOILP as follows

Definition 2.4. Given a MOILP $\min\{Cx \mid x \in \mathcal{X}\}$, we say that the MOILP is unbounded if there exists a vector $\lambda \in \mathbb{R}^p_{\geqq}$ such that for all $l \in \mathbb{R}$ there exists an $\hat{x} \in \mathcal{X}$ for which

$$\lambda C\hat{x} < l$$

Thus, we define a MOILP to be unbounded if there exists a weighted sum scalarization (with non-negative weights), which is unbounded in the single objective sense. From this definition, in combination with Theorem 2.2, the unboundedness of a MOILP can be detected by inspecting whether there exists a weighted sum scalarization whose linear relaxation is unbounded. In that case, the MOILP is either unbounded or infeasible as the MOILP has the same feasible set as any weighted sum scalarization hereof.

When using Algorithm 2.2 for solving the linear relaxation, no weighted sum scalarization is solved directly. However, unboundedness can be detected by inspecting the extreme rays of $\mathcal{Y}_N^{LP} + \mathbb{R}^p_{\geqq}$. Let r be an extreme ray of $\mathcal{Y}_N^{LP} + \mathbb{R}^p_{\geqq}$ such that at least one of its components is strictly positive, and another is strictly negative. When moving along such an extreme ray, because of the conditions on r, at least one objective function value will increase, meanwhile at least another one will decrease. This imply that the weighted sum for which the hypothetical extreme point located infinitely far at the end of the ray is optimal is unbounded, meaning that the MOILP is unbounded in the sense of Definition 2.4.

2.B Problem classes

2.B.1 Production Planning Problem

At each period, $t \in T$, a fixed demand, d_t, for a product is known. This demand must be met from either production in the period, from the inventory or as a combination of produced and stored items. The production in period t is given by x_t while the number of items in inventory at the end of period t is given by s_t. Both x_t and s_t are assumed to be integers.

If at least one unit is produced at period t, a fixed cost is incurred. The variables y_t indicate whether at least one item is produced at time t ($y_t = 1$) or not ($y_t = 0$).

The multi-objective production planning problem (PPP) with p objectives can then be described as the following MOILP:

$$\begin{aligned}
\min\ & \sum_{t=1}^{T}(c_t^k x_t + h_t^k s_t + f_t^k y_t) && \forall k \in \{1, ..., p\}\\
\text{s.t. } & x_t + s_{t-1} = s_t + d_t && \forall t \in \{1, \dots, T\}\\
& x_t \leq M y_t, && \forall t \in \{1, \dots, T\}\\
& s_0 = 0, &&\\
& x_t,\ s_t \in \mathbb{N} && \forall t \in \{1, \dots, T\}\\
& y_t \in \{0, 1\} && \forall t \in \{1, \dots, T\}
\end{aligned}$$

The production costs, storage costs, and fixed costs are given by c_t^k, h_t^k, and f_t^k, respectively. Production and storage costs are generated randomly in the interval $[1, 100]$, and fixed costs in the interval $[1, 2500]$. The demands, d_t, are generated randomly in $[1, 50]$. The total number of variables is given by $3T$, and the number of constraints is $2T$. The parameter M in the indicator constraints is given by $M = \sum_{t=1}^{T} d_t$.

2.B.2 Uncapacitated Facility Location Problem

In this problem, there is a set of l locations where a facility can be opened, and a set of r customers that each have to be assigned to a location. Two decisions have to be made: which locations to open and which customers to assign to which facilities. Both opening a facility and assigning a customer to an open facility induces a cost, and the overall cost has to be minimized. Let $y_j = 1$ if a facility is opened at location j, and $y_j = 0$ otherwise, $\forall j \in \{1, ..., l\}$. Furthermore, let $x_{ij} = 1$ if customer i is assigned to location j, and $x_{ij} = 0$ otherwise, $\forall i \in \{1, ..., r\}, \forall j \in \{1, ..., l\}$.

The multi-objective uncapacitated facility location problem (UFLP) with p objectives can be formulated as the following MOCO problem

$$\begin{aligned}
\min\ & \sum_{i=1}^{r}\sum_{j=1}^{l} c_{ij}^{k} x_{ij} + \sum_{j=1}^{l} f_j^k y_j && \forall k \in \{1,...,p\}\\
\text{s.t. } & \sum_{j=1}^{l} x_{ij} = 1 && \forall i \in \{1,\dots,r\}\\
& x_{ij} \le y_j && \forall i \in \{1,\dots,r\},\ j \in \{1,\dots,l\}\\
& x_{ij} \in \{0,1\} && \forall i \in \{1,\dots,r\},\ j \in \{1,\dots,l\}\\
& y_j \in \{0,1\} && \forall j \in \{1,\dots,l\}
\end{aligned}$$

The cost for assigning customer i to facility j in objective k is given by c_{ij}^k. The assignment costs belongs to the interval $[1, 1000]$ and is generated on the non-dominated part (in minimization) of an hypersphere of dimension p (see Nielsen (2020) for further details). The cost for opening a facility on location j is given by f_j. The fixed opening costs are generated from the interval $[1, 100]$ and, like the assignment costs, these coefficients are generated on the non-dominated part (in minimization) of an hypersphere of dimension p. The number of variables in this problem is $n = l(r + 1)$.

Third Chapter

Branch-and-bound and objective branching with three or more objectives

History: This chapter has been prepared in collaboration with Sune Lauth Gadegaard, Kathrin Klamroth, Anthony Przybylski, and Lars Relund Nielsen. Preliminary results were obtained prior to the PhD in collaboration with Kathrin Klamroth and Anthony Przybylski, and presented at an internal seminar in November 2019 at Aarhus University, Denmark. The work was then further developed, and presented again at the RAMOO workshop in 2020 in Linz, Austria. The latest version, improved by the results from Chapter 2, has been accepted for publication in Computers & Operation Research.

Branch-and-bound and objective branching with three or more objectives

Nicolas Forget*, Sune Lauth Gadegaard*, Kathrin Klamroth**, Lars Relund Nielsen, Anthony Przybylski***

* Department of Economics and Business Economics, School of Business and Social Sciences, Aarhus University, Denmark

** School of Mathematics and Natural Sciences, University of Wuppertal, Germany

*** Faculty of Science and Technology, University of Nantes, France

Abstract

The recent success of bi-objective Branch-and-Bound (B&B) algorithms heavily relies on the efficient computation of upper and lower bound sets. These bound sets are used as a supplement to the classical dominance test to improve the computational time by imposing inequalities derived from (partial) dominance in the objective space. This process is called objective branching since it is mostly applied when generating child nodes. In this paper, we extend the concept of objective branching to multi-objective integer optimization problems with three or more objective functions. Several difficulties arise in this case, as there is no longer a natural order among non-dominated objective vectors when there are three or more objectives. We discuss the general concept of objective branching in any number of dimensions and suggest a merging operation of local upper bounds to avoid the generation of redundant sub-problems. Finally, results from extensive experimental studies on several classes of multi-objective optimization problems are reported.

Keywords: multi-objective combinatorial optimization; multi-objective integer programming; branch & bound; objective branching; bound sets.

3.1 Introduction

In many real-world problems, often more than one objective need to be optimized. Indeed, a decision maker may be interested in minimizing one objective, e.g., operational costs, while at the same time maximizing customer satisfaction. These different objectives are often conflicting, and hence, we cannot realistically expect to find a single solution that optimizes all objectives simultaneously. Thus, a set of trade-off solutions should be produced and, for this purpose, a multi-objective optimization problem must be solved. More precisely, we are interested in generating all rational compromises between the conflicting objectives of *multi-objective integer linear optimization* (*MOILP*) problems, where all variables are integer. In this paper we will consider multi-objective optimization problems with three or more objective functions that must be optimized simultaneously. A particular type of MOILP, called *multi-objective combinatorial optimization* (*MOCO*) problem, has received a specific attention in the literature. Also, although most research has focused on bi-objective MOILP problems, the interest in MOILP problems with more objectives has risen during the last 10-20 years.

The methodology for solving MOILP can be roughly divided into two main groups: *objective space* and *decision space* search algorithms. As the names suggest, the two methodologies work in the space of the objective functions and the space of the decision variables, respectively. A rather large body of literature exists on objective space search algorithms. The objective space search algorithms work by *scalarizing* the MOILP problem and then solving a series of single objective optimization problems, thus utilizing the power of modern commercial integer programming (IP) solvers.

A rather straightforward approach for MOILP problems was proposed by Sylva and Crema (2004). They solve a series of IPs that becomes more and more constrained as non-dominated points are generated. The procedure leads to the solution of exactly $|\mathcal{Y}_N|+1$ single objective IPs where $\mathcal{Y}_N$ is the set of non-dominated objective vectors. Despite its simplicity and low number of IPs solved, the disjunctive nature of the added constraints makes the IPs excessively hard to solve, even after generating only a small subset of the non-dominated points.

During the last 10 years, more advanced objective space search algorithms solving more but easier IPs have been proposed. The interested reader is referred to Ozlen et al. (2014) for an algorithm based on recursion that can handle an arbitrary number of objectives; to Dächert and Klamroth (2015) and Klamroth et al. (2015) for a decomposition into pairwise non-redundant search zones in three and in arbitrary dimensions, respectively. Tamby and Vanderpooten (2021) developed an efficient strategy to enumerate these search zones using ε-constraint scalarizations. Finally, Kirlik and Sayın (2014) and Boland et al. (2017) proposed methods based on ε-constraint scalarizations in combination with dimension reduction for the multi and tri-objective cases, respectively, and Boland and Savelsbergh (2016) proposed the L-shaped search method for problems with three objectives.

One of the main drawbacks of the objective space search algorithms is that an immense amount of information about the search is lost every time a new IP is solved by the black-box solver, as the branching trees created by the solver cannot be directly reused after adding constraints on the objective functions. This problem can be circumvented if the search procedure employs a “one-tree” search strategy where the method searches for efficient solutions in the decision space instead of searching for non-dominated points in the objective space.

One of the first Branch-and-Bound (B&B) based algorithms for multi-objective integer programming problems was developed by Klein and Hannan (1982). The algorithm uses post optimality techniques to solve a series of integer problems all in one tree structure. In the 1980s and 1990s, only a few papers on multi-objective B&B algorithms were published, and the authors have only been able to identify the paper by Kiziltan and Yucaoğlu (1983), in which a general B&B framework is developed. For problem specific procedures based on decision space search algorithms, we refer the reader to Ulungu and Teghem (1995, 1997); Ramos et al. (1998), and Visée et al. (1998).

During the last 20 years, increasing attention has been given to decision space search methods: Mavrotas and Diakoulaki (1998) developed a B&B methodology that even allows for some of the variables to be continuous, and they later refine some parts of the algorithm in Mavrotas and Diakoulaki (2005). The algorithm is applied to the multi-objective, multi-dimensional knapsack problem in Florios et al. (2010). It was later shown by Vincent et al. (2013) that the algorithm by Mavrotas and Diakoulaki (1998) is incorrect as it might return dominated solutions that are considered non-dominated by the algorithm. This problem is remedied for the bi-objective case by Vincent et al. (2013). The mixed-integer case was also explored by Belotti et al. (2013) and Adelgren and Gupte (2022) for the bi-objective case.

The contributions on decision search space algorithms mentioned above all solely rely on variable branching and use a single ideal/utopian point as the lower bound for fathoming nodes in the branching tree, except Vincent et al. (2013) who test additional lower bound sets, such as the linear and convex relaxations, and Belotti et al. (2013) and Adelgren and Gupte (2022) who also rely on the use of the linear relaxation. Sourd and Spanjaard (2008) continue to use branching on single variables as well, but introduce a bounding procedure that is based on a set of points instead of a single ideal point of the branching node: the branching node under scrutiny can be fathomed if a hypersurface separates the set of feasible points in the subproblem from the incumbent set.

This idea is further developed in Stidsen et al. (2014) for the bi-objective case where hyperplanes obtained from solving a weighted sum scalarization of the LP relaxation are used as a lower bound set. Furthermore, Stidsen et al. propose what they call *Pareto branching*, which essentially uses some of the ideas from objective space search algorithms but embed them in a decision space search strategy. The algorithm is further developed in Stidsen and Andersen (2018) where the objective space is partitioned in so-called slices that make parallelization possible in completely non-overlapping subproblems leading to no information sharing between parallel processes.

The concept of Pareto branching is further developed by Parragh and Tricoire (2019) and by Gadegaard et al. (2019) who, independently, propose to partition the objective space into disjoint areas defined by local upper bounds dominated by a lower bound set. Parragh and Tricoire propose to generate the extreme supported points of each node and use those points to generate a lower bound set, whereas Gadegaard et al. use the efficient points of the bi-objective LP relaxation with additional cutting planes as a lower bound set. Parragh and Tricoire (2019) show that their algorithm is particularly efficient compared to objective space search algorithms when the polyhedral description can be significantly improved since in this case, "problem specific" knowledge can be used throughout the algorithm, whereas objective space methods repeatedly call standard IP-solvers that solve the IPs from scratch over and over again. For a recent survey on the components of multi-objective B&B, the reader is referred to Przybylski and Gandibleux (2017).

Recently, attention was given to multi-objective branch-and-bound frameworks. Santis et al. (2020) and Forget, Gadegaard, and Nielsen (2022) independently developed a generic linear-relaxation based multi-objective branch-and-bound framework that can handle three or more objective functions. The main difference lies in the computation of the lower bound set and the problem class solved. Santis et al. (2020) aim at solving mixed-integer convex optimization problems. In their framework, they solve a single-objective linear program for each local upper bound for which it is not known whether it is dominated by the linear relaxation. In case they conclude that the local upper bound is dominated by the lower bound set, the dominance test stops. Otherwise, they generate a cutting plane that may allow them to detect other non-dominated local upper bounds without solving a linear program. This is an improved version of the *implicit lower bound set computation* proposed for the bi-objective case by Gadegaard et al. (2019). In Forget et al. (2022), the focus is on multi-objective integer linear optimization problems. They explicitly compute the linear relaxation by using an outer approximation method (see Benson (1998), and Hamel et al. (2013), Csirmaz (2015), and Löhne and Weißing (2020) for improvements), and accelerate its computation by warm-starting the outer approximation algorithm using the lower bound set from the father node.

However, none of these methods utilize the information from the objective space in order to speed up the algorithm, as it is done in most state-of-the-art bi-objective branch-and-bound frameworks. Hence, in this paper, we aim at extending Pareto branching to problems with three or more objective functions. The main contribution of this paper is fourfold:

1. We highlight difficulties and differences that arise when applying objective branching to problems with an arbitrary number of objective functions, in contrast to the bi-

objective case.

2. We introduce the concept of super local upper bounds in order to limit the computation of redundant LP relaxations and propose an algorithm that computes a uniquely defined set of such super local upper bounds.
3. A set of useful properties for objective space branching is established, and we show that the suggested branching scheme satisfies these properties. Hence, we generalize objective space branching to an arbitrary number of objective functions.
4. The proposed algorithm is evaluated by an extensive computational study based on sets of multi-objective integer linear optimization problems that exhibit a variety of structural properties.

The remainder of the paper is organized as follows. Preliminary definitions and notation for multi-objective optimization are given in Section 3.2. Our multi-objective B&B framework is described in Section 3.3. Section 3.4 outlines the principle of objective branching, presents the difficulties that arise with three objectives and develops a strategy to compute objective branching in the multi-objective case. Finally, experiments are provided in Section 3.5, and a conclusion as well as proposals for further research are given in Section 3.6.

3.2 Definitions and notations

Consider a *multi-objective combinatorial optimization problem* P

$$P: \quad \min\{z(x) = Cx \mid x \in \mathcal{X}\}$$

with *feasible set* $\mathcal{X} = \{x \in \mathbb{N}^n \mid Ax \geqq b\}$ where n is the number of variables, $A \in \mathbb{Z}^{m\times n}$ is a matrix defining the coefficients of the m constraints with right hand side $b \in \mathbb{Z}^m$. The p linear objectives are defined using the matrix $C \in \mathbb{Z}^{p\times n}$ of objective function coefficients. The corresponding image in the *objective space* is $\mathcal{Y} = \{z(x) \mid x \in \mathcal{X}\} := C\mathcal{X}$. Moreover, let P^{LP} denote the *linear relaxation* of P

$$P^{LP}: \quad \min\{z(x) = Cx \mid x \in \mathcal{X}^{LP}\}$$

with feasible set $\mathcal{X}^{LP} = \{x \geqq 0 \mid Ax \geqq b\}$.

Since there are several objective functions, binary relations to compare vectors need to be introduced. Let $y^1, y^2 \in \mathbb{R}^p$, then $y^1 \leqq y^2$ (y^1 *weakly dominates* y^2) if $y^1_k \leq y^2_k$, $\forall k \in \{1, ..., p\}$. Moreover, $y^1 \leqslant y^2$ (y^1 *dominates* y^2) if $y^1 \leqq y^2$ and $y^1 \neq y^2$. Finally, $y^1 < y^2$ (y^1 *strictly dominates* y^2) if $y^1_k < y^2_k$, $\forall k \in \{1, ..., p\}$. Furthermore, for $x \in \mathcal{X}$, we say that x is *efficient* if there is no $x' \in \mathcal{X}$ such that $z(x') \leqslant z(x)$, and x is *weakly efficient* if there is no $x' \in \mathcal{X}$ such that $z(x') < z(x)$. The *set of efficient solutions* is denoted by

$\mathcal{X}_E = \{x \in \mathcal{X} \mid x \text{ is efficient}\}$, and the *set of non-dominated points* is denoted by $\mathcal{Y}_N := C\mathcal{X}_E$. More generally, given a set $\mathcal{S} \in \mathbb{R}^p$, the set of non-dominated points of $\mathcal{S}$ will be denoted by $\mathcal{S}_N = \{s \in \mathcal{S} \mid \nexists s' \in \mathcal{S},\ s' \leqslant s\}$.

3.2.1 Bound sets

Given a $\mathcal{S} \subseteq \mathbb{R}^p$, it is possible to define lower and upper bound sets for $\mathcal{S}_N$. For this purpose, we use the definitions proposed in Ehrgott and Gandibleux (2007). Let $\mathbb{R}^p_{\geqq} := \{y \in \mathbb{R}^p \mid y \geqq 0\}$ and define $\mathbb{R}^p_{\geqslant}$ and $\mathbb{R}^p_{>}$ analogously. Given a set $\mathcal{S} \subseteq \mathbb{R}^p$, we say that $\mathcal{S}$ is $\mathbb{R}^p_{\geqq}$-closed if the set $\mathcal{S} + \mathbb{R}^p_{\geqq}$ is closed, and $\mathbb{R}^p_{\geqq}$-bounded if there exists $s \in \mathbb{R}^p$ such that $\mathcal{S} \subset s + \mathbb{R}^p_{\geqq}$. Let cl(.) denote the closure operator.

Definition 3.1. Consider a set of points $\mathcal{S} \subseteq \mathbb{R}^p$.

- A lower bound set $\mathcal{L} \subseteq \mathbb{R}^p$ of $\mathcal{S}_N$ is an $\mathbb{R}^p_{\geqq}$-closed and $\mathbb{R}^p_{\geqq}$-bounded set such that $\mathcal{S}_N \subset (\mathcal{L} + \mathbb{R}^p_{\geqq})$ and $\mathcal{L} = \mathcal{L}_N$.
- An upper bound set $\mathcal{U}$ of $\mathcal{S}_N$ is an $\mathbb{R}^p_{\geqq}$-closed and $\mathbb{R}^p_{\geqq}$-bounded set such that $\mathcal{S}_N \subset \text{cl}[\mathbb{R}^p \backslash (\mathcal{U} + \mathbb{R}^p_{\geqq})]$ and $\mathcal{U} = \mathcal{U}_N$.

One specific lower bound set and one specific upper bound set of $\mathcal{Y}_N$ are the ideal point y^I given by

$$y^I_k = \min_{y \in \mathcal{Y}} \{y_k\}, \forall k \in \{1, ..., p\},$$

and the nadir point y^N defined by

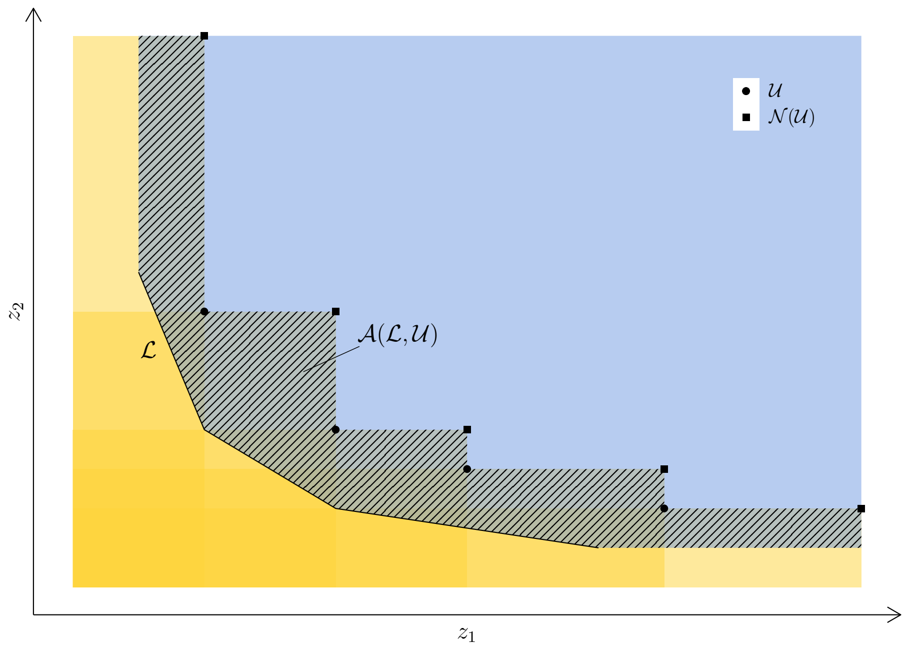
$$y^N_k = \max_{y \in \mathcal{Y}_N} \{y_k\}, \forall k \in \{1, ..., p\}.$$

Given two sets $\mathcal{S}^1$ and $\mathcal{S}^2$, we say that $\mathcal{S}^1$ is *fully weakly dominated* by $\mathcal{S}^2$ if $\forall s^1 \in \mathcal{S}^1$, $\exists s^2 \in \mathcal{S}^2$ such that $s^2 \leqq s^1$. Furthermore, we say that $\mathcal{S}^1$ is *partially dominated* by $\mathcal{S}^2$ if $\mathcal{S}^1$ is not fully weakly dominated by $\mathcal{S}^2$ and if there exists at least one pair of points (s^1, s^2) such that $s^1 \in \mathcal{S}^1$, $s^2 \in \mathcal{S}^2$, and $s^2 \leqslant s^1$.

For the purpose of readability, in the remainder of this paper, we will consider that a lower bound set *for a problem* P' is the equivalent of a lower bound set *for the set of non-dominated points of* P'.

3.2.2 Search region and local upper bounds

Given an upper bound set $\mathcal{U}$ for P, i.e., an upper bound set of $\mathcal{Y}_N$ and a lower bound set $\mathcal{L}$ of P or a subproblem of P, it is possible to determine the *search region.* The search region defines the region of the objective space where feasible (for P or a subproblem of P)



● $\mathcal{U}$
■ $\mathcal{N}(\mathcal{U})$
$\mathcal{A}(\mathcal{L},\mathcal{U})$
$\mathcal{L}$
z_2
z_1

Figure 3.1: Given a lower bound set $\mathcal{L}$ (the line segments) and an upper bound set $\mathcal{U}$, the search region $\mathcal{A}(\mathcal{L},\mathcal{U})$ is given by the closure of the hatched area.

non-dominated (for P) points are located. For this purpose, the theory from Klamroth et al. (2015) will be used and be applied to our specific framework.

First, note that due to Definition 3.1, $\mathcal{Y}_N \subset \mathcal{A}(\mathcal{U}) := \mathrm{cl}(\mathbb{R}^p \backslash (\mathcal{U} + \mathbb{R}^p_{\geqq}))$ (closure of the hatched and yellow areas in Figure 3.1). This region corresponds to the part of the objective space that is not dominated by any point of the upper bound set. An alternative description can be used by introducing the concept of *local upper bounds*.

Definition 3.2. Let $\mathcal{C}(u) = u - \mathbb{R}^p_{\geqq}$ be the *search cone* given the point $u \in \mathbb{R}^p$. The set of local upper bounds with respect to $\mathcal{U}$, $\mathcal{N}(\mathcal{U})$ is a unique discrete set of points in $\mathbb{R}^p$ satisfying

1. $\mathcal{A}(\mathcal{U}) = \bigcup\limits_{u \in \mathcal{N}(\mathcal{U})} \mathcal{C}(u)$ and
2. $\mathcal{N}(\mathcal{U})$ is minimal, i.e., there is no $u^1, u^2 \in \mathcal{N}(\mathcal{U})$, $u^1 \neq u^2$, such that $\mathcal{C}(u^1) \subseteq \mathcal{C}(u^2)$.

From Definition 3.2 it follows that $\mathcal{Y}_N \subset \mathcal{A}(\mathcal{U}) = \mathcal{N}(\mathcal{U}) - \mathbb{R}^p_{\geqq}$ and due to Definition 3.1 we have $\mathcal{Y}_N \subset \mathcal{L} + \mathbb{R}^p_{\geqq}$ (hatched and blue areas in Figure 3.1). Thus, $\mathcal{Y}_N \subset (\mathcal{L} + \mathbb{R}^p_{\geqq}) \cap (\mathcal{N}(\mathcal{U}) - \mathbb{R}^p_{\geqq})$ (closure of hatched area in Figure 3.1). In the remainder of the paper, we assume that $\mathcal{U} \subseteq \mathcal{Y}$, thus the upper boundary of $\mathcal{A}(\mathcal{U})$ is dominated, and hence we define the *search region* as $\mathcal{A}(\mathcal{L},\mathcal{U}) = (\mathcal{L} + \mathbb{R}^p_{\geqq}) \cap (\mathcal{N}(\mathcal{U}) - \mathbb{R}^p_{>})$.

3.3 General multi-objective Branch-and-Bound framework

In this section, a multi-objective Branch-and-Bound (B&B) framework will be presented. The aim is to compute the set of non-dominated points $\mathcal{Y}_N$ and a corresponding *minimal complete* set of efficient solutions. The framework is mainly based on the framework of Forget et al. (2022), and a general outline is recalled in this section for the sake of completeness. The theory and methodology presented in this paper can be applied to any linear problem whose non-dominated set is made of a finite number of points. This includes for instance mixed-integer problems where continuous variables are present in at most one objective. However, we restrict ourselves to problems with only integer variables to avoid an additional layer of notation.

The basic idea of the B&B algorithm is to divide an initial problem P that cannot be solved easily into less complex disjoint subproblems. The algorithm manages a tree data structure in which each problem (subproblem) is stored as a node. Given a node η, its corresponding problem will be $P(\eta)$, and its feasible set will be $\mathcal{X}(\eta)$. The linear relaxation of the problem in question will be denoted by $P^{LP}(\eta)$, and its feasible set will be $\mathcal{X}^{LP}(\eta)$. The nodes containing the subproblems created from $P(\eta)$ will be stored as child nodes of η. The tree is initialized by creating the root node, that contains the initial problem P. The upper bound set (on the set of non-dominated points of the initial problem P) $\mathcal{U}$ is initialized as an empty set. It may be initialized with feasible solutions if some are known prior to the executing the algorithm, but this will not be the case in this paper. The outline of the multi-objective B&B is given in Algorithm 3.1.

In Step 1, we select a node η that will be explored. The depth-first and the breadth-first strategies are the most frequently used strategies for this purpose in the literature on multi-objective B&B (see Przybylski and Gandibleux (2017)). In practice, there is no best strategy among the two. In fact, it appears to be very problem specific. For example, Visée et al. (1998) and Vincent et al. (2013) show that depth-first performs better on their set of instances, whereas Parragh and Tricoire (2019) and Forget et al. (2022) show that for some problem classes breadth-first performs the best.

At each node η, the linear relaxation $P^{LP}(\eta)$ will be solved and yields a lower bound set $\mathcal{L}(\eta)$ for $P(\eta)$, the problem included in node η (Step 2). In the multi-objective case, this corresponds to the non-dominated part of a convex polytope of dimension at most p. In order to compute the linear relaxation, the algorithm of Forget et al. (2022) will be used. A description of $\mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}$ as a collection of hyperplanes and extreme points is then obtained, which will be of interest for Steps 3, 4, and 5.

In Section 3.4, we show that it is necessary to compute the dominance test for each local

```
Input: $P$: A multi-objective combinatorial optimization problem $P$.
Step 0: Initialize the B&B tree with a list of non-explored nodes $\Gamma$ containing the initial problem $P$.
Step 1: If $\Gamma = \emptyset$, go to Step 6. Otherwise, choose a node $\eta \in \Gamma$, and remove it from $\Gamma$.
Step 2: Compute a lower bound set $\mathcal{L}(\eta)$ for $P(\eta)$.
Step 3: If $\eta$ can be fathomed, go to Step 1. If possible, update the upper bound set $\mathcal{U}$.
Step 4: Split $P(\eta)$ by branching in the objective space. New subproblems are obtained.
Step 5: Split each subproblem by branching in the decision space. Create a node $\eta'$ for each sub-subproblem and add it to $\Gamma$. Go to Step 1.
Step 6: End of the Algorithm. Return the upper bound set.
Output: The set of non-dominated points $\mathcal{Y}_N$.
```

Algorithm 3.1: Multi-objective B&B algorithm

upper bound, i.e., we cannot stop when we find one local upper bound dominated by the lower bound set. Furthermore, in Section 3.4.2, many dominance tests will be performed in order to compute objective branching. The framework from Santis et al. (2020) involves solving exactly one linear program per point dominated by the lower bound set, and at most one for the ones that are not dominated, which would result in a very expensive procedure. Similar tests were performed in Gadegaard et al. (2019) for the bi-objective case, and the authors show that computing what they called an explicit lower bound set was significantly more efficient when coupled with objective branching, which is in line with the results in Forget et al. (2022).

Step 3 uses three procedures to fathom η. First, if $P^{LP}(\eta)$ is infeasible, $P(\eta)$ is also infeasible (because $\mathcal{X}(\eta) \subseteq \mathcal{X}^{LP}(\eta)$). In this case, the node is fathomed by infeasibility. Second, if the lower bound set consists of a unique extreme point y with a pre-image feasible for $P(\eta)$, it can be concluded that any solution found in a subproblem of $P(\eta)$ will have an objective vector (weakly) dominated by y. Consequently, the subproblem requires no further examination, and the corresponding node η is fathomed by optimality. The point y is eventually added to the upper bound set $\mathcal{U}$, if it is not dominated by any point of the upper bound set. In this case, the points of the upper bound set that are dominated by y are deleted as well.

Finally, if none of the two mentioned cases occur, a third fathoming test is used: the dominance test. The purpose of this step is to detect whether the search region at node η, denoted by $\mathcal{A}(\mathcal{L}(\eta),\mathcal{U})$, is empty. Indeed, if this is true, this implies that no non-dominated point feasible for the initial problem P can be found in the subproblem $P(\eta)$ and thus, it requires no further examination. In this case, η is fathomed by dominance. In order to check that, the dominance test from Forget et al. (2022) will be used, and it is recalled in Proposition 3.1 and Lemma 3.1 (we refer the reader to this paper for the proofs). Note that as both the variables in feasible solutions and the objective coefficients only take integer values, the feasible objective vectors will always have integer components as well. Furthermore, since a local upper bound $u \in \mathcal{N}(\mathcal{U})$ by definition is dominated by some points in the upper bound set, there is no need to search for a new point that has the same components as u. Hence, each local upper bound $u \in \mathcal{N}(\mathcal{U})$ is replaced by a shifted local upper bound $\bar{u} = u - e$, where $e = (1, ..., 1) \in \mathbb{R}^p$.

Proposition 3.1. *The search region $\mathcal{A}(\mathcal{L}(\eta), \mathcal{U})$ is empty if $(\mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}) \cap \mathcal{N}(\mathcal{U}) = \emptyset$.*

By using Proposition 3.1, it can be concluded that η can be fathomed by dominance if there is no $u \in \mathcal{N}(\mathcal{U})$ such that $u \in \mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}$. In order to check the condition, the hyperplane representation of $\mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}$ will be used. Let $\mathcal{H}^1, ..., \mathcal{H}^F$ be these hyperplanes. We then have $\mathcal{H}^i = \{y \in \mathbb{R}^p \mid n^{iT} \cdot y = d^i\}$, where $n^i \in \mathbb{R}^p$ is the normal vector of $\mathcal{H}^i$ and $d^i \in \mathbb{R}$. The polytope $\mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}$ can be defined as an intersection of closed half-spaces defined by $\mathcal{H}^1, ..., \mathcal{H}^F$, i.e., we have

$$\mathcal{L}(\eta) + \mathbb{R}^p_{\geqq} = \bigcap_{i=1}^{F} \{y \in \mathbb{R}^p \mid n^{iT} \cdot y \geq d^i\}.$$

Hence, a point $\tilde{y} \in \mathbb{R}^p$ is located in $\mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}$ if for each $i \in \{1, ..., F\}$, $n^{iT} \cdot \tilde{y} \geq d^i$ holds true. This leads to Lemma 3.1.

Lemma 3.1. *Let $\mathcal{H}^1, ..., \mathcal{H}^F$ be the hyperplane representation of $\mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}$, where $\mathcal{H}^i = \{y \in \mathbb{R}^p \mid n^{iT} \cdot y = d^i\}$, $n^i \in \mathbb{R}^p$ is the normal vector of $\mathcal{H}^i$, and $d^i \in \mathbb{R}$. The node η can be fathomed by dominance if for each $\bar{u} \in \mathcal{N}(\mathcal{U})$, there exists $i \in \{1, ..., F\}$ such that $n^{iT} \cdot \bar{u} \leq d^i$.*

If η could not be fathomed by any of the three procedures described above, the upper bound set $\mathcal{U}$ is updated by considering the extreme points of the lower bound set with a pre-image feasible for $P(\eta)$, if any such exists. Let y be such a point, y is added to $\mathcal{U}$ if there is no $u \in \mathcal{U}$ such that $u \leqslant y$. Furthermore, all points $u \in \mathcal{U}$ such that $y \leqslant u$ are deleted from $\mathcal{U}$. Each time the upper bound set is updated, the set of local upper bounds $\mathcal{N}(\mathcal{U})$ is also updated using the procedure proposed in Klamroth et al. (2015).

Finally, if the node η cannot be fathomed, $P(\eta)$ will be divided into disjoint subproblems. There are two ways to create subproblems: in the objective space and in the decision space. The creation of subproblems in the objective space (Step 4) for an arbitrary number of objectives p is the main contribution of this paper. It has been shown to be very efficient for $p = 2$ (see, e.g., Parragh and Tricoire (2019), Gadegaard et al. (2019), Stidsen and Andersen

(2018)). Its extension to more objectives is discussed in Section 3.4, and its impact on a multi-objective B&B algorithm is discussed in Section 3.5.

In decision space, the subproblems are created by selecting a free variable x_i, $i \in \{1, ..., n\}$, and if relevant, a branching value $t \in \mathbb{N}$ (Step 5). Then, either constraint $x_i \leq t$ or constraint $x_i \geq t+1$ is added to the subproblem. Note that in case x_i is binary, i.e. $x_i \in \{0,1\}$, this is equivalent to fixing x_i to 0 or 1. Hence, at a node η, the problem $P(\eta)$ will be split into subproblems $P(\eta_0)$ and $P(\eta_1)$ such that $\mathcal{X}(\eta_0) = \{x \in \mathcal{X}(\eta) \mid x_i \leq t\}$ and $\mathcal{X}(\eta_1) = \{x \in \mathcal{X}(\eta) \mid x_i \geq t+1\}$. Consequently, a variable x_i is said to be free at node η if it is not fixed to any specific value due to the branching constraints. The choice of the free variable to branch on and, if relevant, of the branching value v at a given node is discussed in Section 3.5.

3.4 Objective branching

The principle of objective branching is to apply a branching rule in the objective space in addition to branching in the decision space. In the bi-objective case, there are several ways to perform objective branching, for instance slicing (Stidsen et al., 2014; Stidsen and Andersen, 2018). In this study, the focus will be on *objective branching* as defined in Stidsen et al. (2014), Gadegaard et al. (2019), and Parragh and Tricoire (2019) (alternatively called Pareto branching or Extended Pareto branching in some of these references). The reason for this choice is that the way it is defined naturally extends to an arbitrary number of objectives p. The definition is given in Definition 3.3.

Definition 3.3. Let $P(\eta)$ be the problem at a node η of the Branch-and-Bound (B&B) tree and $\bar{y} \in \mathbb{R}^p$. It is said that *objective branching is applied on* $\bar{y}$ if the subproblem $P(\eta, \bar{y}) \coloneqq \min\{Cx \mid x \in \mathcal{X}(\eta),\ z(x) \leqq \bar{y}\}$ is created.

Note that in this definition of objective branching, no constraint in the form $z_k(x) \geqq \bar{z}_k$ is used because such a constraint may result in increasing cpu time (Stidsen et al., 2014).

Now consider the situation depicted in Figure 3.2 for a node η in the branching tree that cannot be fathomed. One can observe that even though the lower bound set $\mathcal{L}(\eta)$ is not fully dominated by the upper bound set $\mathcal{U}$, it is still partially dominated, i.e., there exist some $l \in \mathcal{L}(\eta)$ such that $u \leqslant l$, with $u \in \mathcal{U}$. Hence, there is no need to spend computational efforts (e.g., computing lower bound sets, searching for new integer points and so forth) to explore the objective space where the lower bound set is already dominated. The purpose of objective branching is to discard these regions by creating subproblems in the objective space. For example, in Figure 3.2, three subproblems are created by applying objective branching on the points y^1, y^2 and y^3, resulting in the subproblems $P(\eta, y^1)$, $P(\eta, y^2)$, and $P(\eta, y^3)$

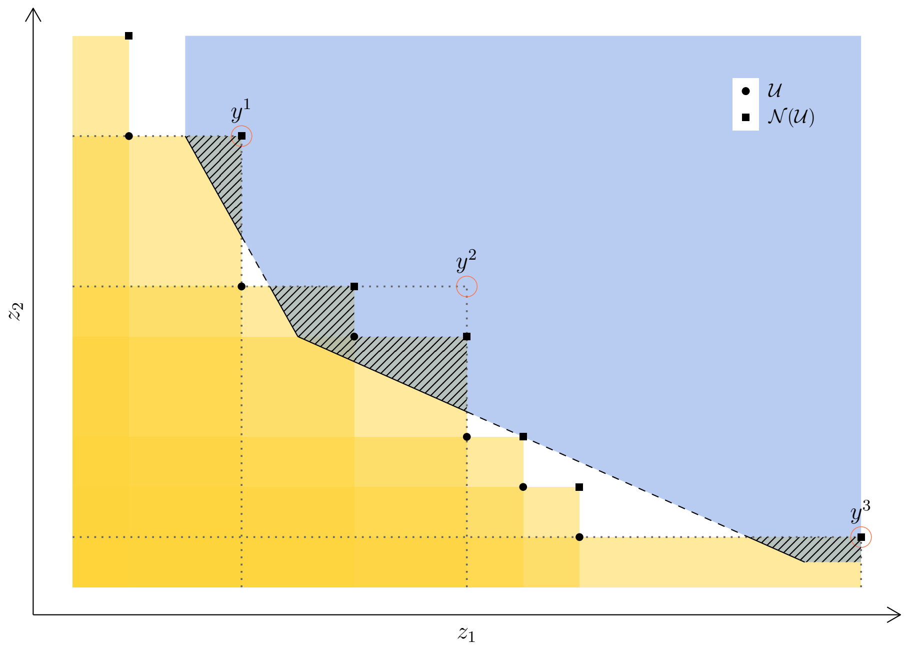


Figure 3.2: A lower bound set (solid and dashed lines) partially dominated by the upper bound set. The dominated area of the lower bound set is represented by the dashed lines. Three disjoint subproblems are created in the objective space by applying objective branching on y^1, y^2 and y^3, represented by the red circles. The constraints added when applying objective branching are represented by the dotted lines. A new non-dominated point feasible for the corresponding problem can only be found in one of these subproblems (dotted areas).

respectively. The corresponding constraints are depicted with dotted lines in Figure 3.2. One can observe that by creating these three subproblems, the parts of the objective space that are already known to be dominated are not included in any of those subproblems and thus they will not be explored in the sub-tree starting from node η.

We need to identify a set of desirable properties to find the points of the objective space that are interesting candidates for objective branching.

Property 3.1. *Let $\mathcal{A}(\mathcal{L}(\eta, s),\mathcal{U})$ denote the search area in problem $P(\eta, s)$ ($\mathcal{L}(\eta, s)$ is the lower bound set of problem $P(\eta, s)$) and ρ be the number of subproblems created in the objective space at node η. Desirable properties for objective branching:*

3.1a) inclusiveness: $\mathcal{A}(\mathcal{L}(\eta),\mathcal{U}) \subseteq \bigcup_{i=1}^{\rho} \mathcal{A}(\mathcal{L}(\eta, s^i),\mathcal{U})$

3.1b) sparsity: $\bigcap_{i=1}^{\rho} \mathcal{A}(\mathcal{L}(\eta, s^i),\mathcal{U}) = \emptyset$

3.1c) tightness: as much dominated area as possible is discarded

Property 3.1a states that each point of the search area at node η should be included in at least one of the subproblems created. In this sense, objective branching should be inclusive.

If this is not satisfied, then a non-dominated feasible point might not be found, as it is not included in any of the subproblems, and therefore the output of the B&B algorithm ($\mathcal{Y}_N$) may not be correct.

Property 3.1b states that the subproblems created in the objective space should be disjoint. This property ensures that a non-dominated feasible point cannot be found several times in the tree, effectively avoiding redundancies.

Property 3.1c states that as many subproblems as possible should be created. This property implies that it is preferable to create two small subproblems instead of one large one, if possible. In other words, as much dominated area as possible should be discarded.

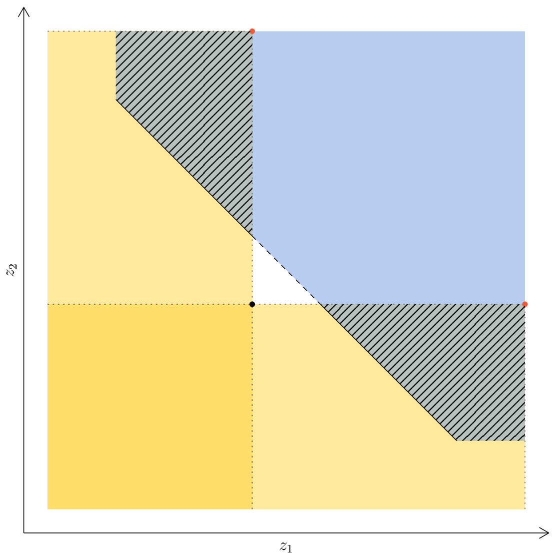
Note that Property 3.1a is crucial for the exactness of the algorithm, whereas Property 3.1b and Property 3.1c are not. Instead, they are here to control the behavior of the algorithm by avoiding redundancies and ensuring tight bounds. Property 3.1a and Property 3.1b are the main focus in the paper, but Property 3.1c will be briefly discussed in the rest of this section and in Section 5.

3.4.1 Complications of going from two to three objectives

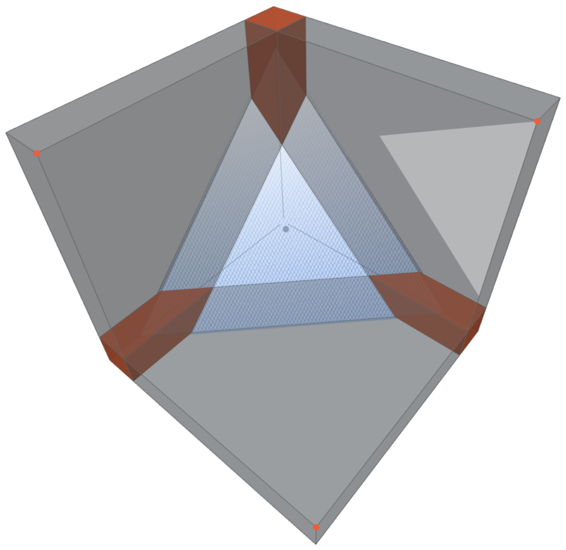
Several approaches have been developed in the literature for identifying the subproblems in the objective space. Stidsen et al. (2014) were the first to propose an approach, which was later improved in Gadegaard et al. (2019). Since they compute a relaxation of the weighted sum scalarization at each node (and obtain a point y, resulting from the single-objective linear program solved in the scalarization), the authors propose to compute objective branching only when there already exists an integer solution dominating y. Such a situation is depicted in Figure 3.3a. The lower bound set consists of a unique hyperplane, and the upper bound set contains a single point that partially dominates the lower bound set. It is possible to create two subproblems that satisfy Property 3.1 by applying objective branching on its local upper bounds.

Exactly the same situation is depicted with three objectives in Figure 3.3b. The lower bound set is given by the blue hyperplane, and the upper bound set is defined by a unique point (in black). The dominated part of the lower bound set is the blue area in the middle. If objective branching is applied on the local upper bounds (red points) as in the bi-objective case, the subproblems created will have redundancies (each subproblem defines a grey search area). Every point in the brown areas is included in the search area of more than one subproblem and thus, it will not satisfy Property 3.1b.

One might infer from Figure 3a that objective branching can be applied whenever the lower bound set is split into several connected components, an inference made by Parragh and Tricoire (2019). They keep track of these connected components and apply objective branching such that each one of them is included in exactly one subproblem. For example,



(a) Bi-objective case: The dominated part of the lower bound set is represented by the dashed line. When objective branching is applied on each local upper bound, the hatched search areas satisfy Property 3.1.

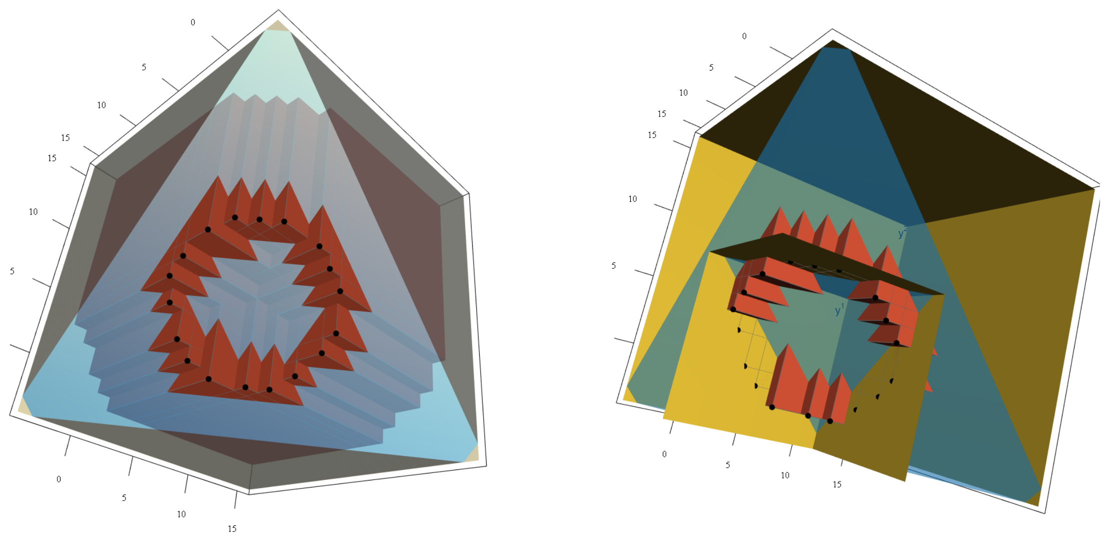


(b) Three-objective case: The dominated part of the lower bound set is the blue area in the middle (view from above). When objective branching is applied on each local upper bound, there are redundancies between the subproblems (each subproblem defines a grey search area). Every point in the brown areas is included in the search area of more than one subproblem.

Figure 3.3: Objective branching given a single point in the upper bound set (black) and a lower bound set consisting of a single hyperplane. Objective branching is applied on each local upper bound point (red). An interactive plot of Figure 3.3b can be seen in Forget et al. (2020b).

in Figure 3.2, there are three non-dominated connected components in the lower bound set, and objective branching is applied on each one.

However, having the lower bound set split is not a sufficient condition for applying objective branching with the desirable properties in the three-objective case. For example, in Figure 3.4, the lower bound set consists of a unique facet. It is partially dominated by the upper bound set (points in black). In the bi-objective case, depicted in Figure 3.3a, it can be observed that there are two non-dominated connected components. However, as seen in Figure 3.3b, applying objective branching on each component like in the bi-objective case will lead to redundancies. In the figure, the two points on which objective branching is applied are represented by the two squares, and the objective branching constraints are given by the two cones starting from these points, and we note that one of the sub-problems is fully included in the other sub-problem.



(a) Lower bound set (hyperplane) partially dominated by the upper bound set (black points with red dominance cones). Note the two disjoint search areas above the hyperplane.

(b) Objective branching applied to y^1 and y^2. One subproblem is fully included in the other subproblem.

Figure 3.4: An example of applying objective branching on two disjoint search areas (view from below). Interactive plots of Figures 3.4a and 3.4b can be seen in Forget et al. (2020b)

It is, however, still possible to apply objective branching in some cases, and a way to detect such cases and to compute the corresponding subproblems is discussed in the next section.

3.4.2 Objective branching in the multi-objective case

The approach developed in this paper identifies points of the objective space such that if objective branching is applied on those points, the subproblems obtained satisfy Property 3.1.

The strategy is based on a merging operation of redundant subproblems, which are defined by the local upper bounds dominated by the lower bound set $\mathcal{L}(\eta)$. Only the dominated local upper bounds are considered since the cone $\mathcal{C}(u)$ of a local upper bound u that is not dominated by $\mathcal{L}(\eta)$ cannot contain any new point feasible for the subproblem $P(\eta)$. Thus, there is no need to search for any feasible point in this area. Hence, these cones are discarded and only the dominated local upper bounds are kept for the merging operation.

Thus, at node η, a set of dominated local upper bounds

$$\mathcal{D}(\eta) = \{u \in \mathcal{N}(U) \mid \exists\, l \in \mathcal{L}(\eta),\ l \leqslant u\}$$

is obtained, and the subproblems will be computed with this set as input. Note that in order to obtain $\mathcal{D}(\eta)$, the algorithm has to check whether each local upper bound is dominated at each node or not, as it requires to know the status of each local upper bound to compute the sub-problems. This is a difference with the classical branch-and-bound framework, where the dominance test can be stopped as soon as a dominated local upper bound is found.

Merging operations on local upper bounds

As explained in Section 3.4.1, one of the most challenging difficulties of objective branching in the multi-objective case is to create subproblems that are pairwise disjoint. Let $s^1, s^2 \in \mathbb{R}^p$ denote two points on which objective branching will be applied. In order to detect whether the subproblems $P(\eta, s^1)$ and $P(\eta, s^2)$ have redundancies, the notion of an *intersection point*, defined in Definition 3.4, will be used.

Definition 3.4. Let $s^1, s^2 \in \mathbb{R}^p$ be two points of the objective space. The intersection point s^I of s^1 and s^2 is the point that yields $s^I_k = \min(s^1_k, s^2_k)$, $\forall k \in \{1, ..., p\}$. The cone $\mathcal{C}(s^I)$ will be called *the intersection cone.*

In particular, $\mathcal{C}(s^I) = \mathcal{C}(s^1) \cap \mathcal{C}(s^2)$. Hence, the intersection cone contains all the points of the objective space that are contained in both $P(\eta, s^1)$ and $P(\eta, s^2)$. Thus, if the intersection point s^I is dominated by $\mathcal{L}(\eta)$, there may exist feasible points that are in $\mathcal{C}(s^I)$, i.e., included in both $C(s^1)$ and $C(s^2)$ and consequently, the subproblems created from s^1 and s^2 are not disjoint.

In this case, the subproblems will be merged. For this purpose, the concept of *super local upper bounds* is now introduced and defined in Definition 3.5. They can be seen as merged local upper bounds.

Definition 3.5. Consider a set of local upper bounds $u^1, ..., u^h \in \mathcal{D}(\eta)$, with $h \in \mathbb{N}\backslash\{0\}$. The point $s \in \mathbb{R}^p$ is a super local upper bound of the local upper bounds $u^1, ..., u^h$ if $s_k = \max\limits_{i \in \{1,...,h\}} u^i_k$, $\forall k \in \{1, ..., p\}$. Furthermore, if a local upper bound $u \in \mathcal{D}(\eta)$ satisfies $u \subset \mathcal{C}(s)$, it is said that u is contained in s.

A super local upper bound can be seen as the nadir point of the set of points $\{u^1, ..., u^h\}$ as well. For each super local upper bound s, we define a set $\mathcal{D}(\eta, s) = \{u \in \mathcal{D}(\eta) \mid u \leqq s\}$ of local upper bounds contained in s at node η.

To conclude, given two local upper bounds $u^1, u^2 \in \mathcal{D}(\eta)$, the goal is to know whether it is possible to apply objective branching on u^1 and u^2 with the desirable properties, or if only one large subproblem should be considered by applying objective branching on a super local upper bound s that contains u^1 and u^2. As explained previously, u^1 and u^2 will be merged if their intersection point u^I is dominated by the current lower bound set $\mathcal{L}(\eta)$. The corresponding super local upper bound obtained, $s \in \mathbb{R}^p$, is defined by $s_k = \max(u^1_k, u^2_k)$. All the reasoning presented here can also be used to merge two super local upper bounds, or a local upper bound and a super local upper bound.

Desirable properties of the set of super local upper bounds

At each node η, a set of super local upper bounds $\mathcal{S}$ with particular properties will be constructed. Then, for each $s \in \mathcal{S}$, the subproblem $P(\eta, s)$ will be created. In order for these problems to satisfy Property 3.1, a set of desirable properties for $\mathcal{S}$ is defined here.

Property 3.2. *The desirable properties of a set of super local upper bounds are the following:*

3.2a) $\forall u \in \mathcal{D}(\eta)$, $\exists s \in \mathcal{S}$ *such that* u *is contained in* s, *i.e.,* $u \in \mathcal{D}(\eta, s)$.

3.2b) If $|\mathcal{S}| \geq 2$, $\forall s^1, s^2 \in \mathcal{S}$, $s^1 \neq s^2$, *the intersection point* s^I *of* s^1 *and* s^2 *is not dominated by the lower bound set* $\mathcal{L}(\eta)$.

3.2c) $\forall s \in \mathcal{S}$, $\forall \epsilon \geqslant 0$, $\exists u \in \mathcal{D}(\eta, s)$ *such that* $u \notin \mathcal{D}(\eta, s - \epsilon)$.

3.2d) The size of $\mathcal{S}$ *is maximal, i.e., we have the maximum number of super local upper bounds in S that satisfy Property 3.1a, Property 3.1b, and Property 3.1c.*

Property 3.2a implies that each dominated local upper bound is merged in at least one super local upper bound, or is a super local upper bound itself. For the objective branching, this implies that no solution is overlooked. Indeed, all the areas in which non-dominated points can be found are included in a super local upper bound and thus, Property 3.1a is satisfied.

Property 3.2b states that it is not possible to merge two or more super local upper bounds of $\mathcal{S}$ with respect to the rule used. If two super local upper bounds do not respect this property, this means that the two corresponding subproblems have an intersection point dominated by the lower bound set and therefore have redundancies, which is the situation that should be avoided. This property ensures that Property 3.1b is satisfied.

Property 3.2c guarantees that the super local upper bounds are as tight as possible and cannot be moved down, i.e., moved in a direction $-\epsilon^T = (-\epsilon_1, ..., -\epsilon_p)$ where $\epsilon \geqslant 0$ is arbitrarily small, without losing at least one of the local upper bounds u it contains. This ensures that as much of the dominated region as possible is discarded in the sub-problems created with objective branching. Hence, this ensures that Property 3.1c is satisfied.

Property 3.2d says that it is not possible to split a super local upper bound $s \in \mathcal{S}$ into several smaller super local upper bounds (i.e., contained in $\mathcal{C}(s)$) without losing one of the previous properties. Without this property, two subproblems could be created instead of one and thus, more of the dominated region could be discarded. Hence, this property has to be

```
1: Input :
2: $\mathcal{L}(\eta)$ : a lower bound set for node $\eta$
3: $\mathcal{D}(\eta)$ : set of local upper bounds dominated by $\mathcal{L}(\eta)$
4: Algorithm :
5: $\mathcal{S} \leftarrow \mathcal{D}(\eta)$
6: while $\exists s^1, s^2 \in \mathcal{S}$ such that their intersection point $s^I$ is dominated by $\mathcal{L}(\eta)$ do
7:     $\mathcal{S} \leftarrow \mathcal{S} \backslash \{s^1, s^2\}$
8:     $s \leftarrow \texttt{Merge}(s^1, s^2)$
9:     $\mathcal{S} \leftarrow \mathcal{S} \cup \{s\}$
10: end while
11: return $\mathcal{S}$
12: Output :
13: $\mathcal{S}$ : set of super local upper bounds
```

Algorithm 3.2: Computation of the set of super local upper bounds

verified to ensure, once again, that as much of the dominated region as possible is discarded, and that it maintains Property 3.1c.

Consequently, obtaining a set $\mathcal{S}$ of super local upper bounds that satisfies Property 3.2 ensures that as much dominated region as possible is discarded while still satisfying the conditions established in Property 3.1.

An algorithm to compute a set of super local upper bounds

In this paragraph we describe the algorithm used to compute a set of super local upper bounds $\mathcal{S}$ satisfying Property 3.2 and show its correctness. In order to work, the algorithm needs the lower bound set $\mathcal{L}(\eta)$ of the node η and the corresponding set of dominated local upper bounds $\mathcal{D}(\eta)$ as input. The algorithm is described in Algorithm 3.2. The function $\texttt{Merge}(s^1, s^2)$ simply merges s^1 and s^2 as presented in Section 3.4.2.

Theorem 3.1. *Algorithm 3.2 computes the set of super local upper bounds.*

Proof. It will be shown that the output $\mathcal{S}$ of this algorithm satisfies Property 3.2.

First, it is important to note that the order in which the (super) local upper bounds are merged does not affect the result. Let s^1, s^2 be two (super) local upper bounds which have to be merged due to Property Property 3.2b. All distinct super local upper bounds s'^1, s'^2 containing s^1 and s^2 respectively have to be merged as their intersection point is, by construction, dominated by the intersection point of s^1 and s^2, itself dominated by the lower bound set.

The set $\mathcal{S}$ is initialized with the set $\mathcal{D}(\eta)$. Furthermore, each time a local upper bound u is deleted from $\mathcal{S}$, a super local upper bound that contains u is created. Ultimately, each dominated local upper bound is included in a super local upper bound, and thus Property 3.2a is satisfied.

Algorithm 3.2 stops when there is no $s^1, s^2 \in \mathcal{S}$ such that their intersection point is dominated by the lower bound set $\mathcal{L}(\eta)$. Hence, by construction, Property 3.2b is satisfied.

A super local upper bound can be defined as a nadir point of some dominated local upper bounds (Definition 3.5). This means that each component (i.e., the value for each objective) of a super local upper bound has the same value as one of the local upper bounds contained in the super local upper bound. This implies that it is not possible to reduce the value of a super local upper bound by $\epsilon \geqslant 0$. Otherwise, it would lose one of the local upper bounds it contains. Hence, by construction of the function `Merge`, Property 3.2c is satisfied.

In order to violate the Property 3.2d, two super local upper bounds that should not be merged would have been merged during the computation. However, this never happens since two super local upper bounds are merged only when their intersection point is dominated by the lower bound set. In other words, they are only merged when the merge respects the rule used. □

In Theorem 3.2, we establish the complexity of Algorithm 3.2. To that end, let $f(\mathcal{L}(\eta))$ be the complexity of checking whether a point is dominated by the lower bound set $\mathcal{L}(\eta)$.

Theorem 3.2. *Algorithm 3.2 runs in* $O(|\mathcal{D}(\eta)|^3 \cdot f(\mathcal{L}(\eta)))$.

Proof. If no pair of super local upper bounds is merged during the main loop, then the algorithm stops. When two super local upper bounds are merged, they are deleted, and a single super local upper bound is constructed instead. At each step, the size of $\mathcal{S}$ is therefore reduced by 1. Thus, at most $|\mathcal{D}(\eta)| - 1$ iterations of the main loop occur.

Each time the algorithm enters its main loop, it needs to identify a pair to merge. For this purpose, a simple pairwise comparison of each element of $\mathcal{S}_t$ can be done, where $\mathcal{S}_t$ is the set $\mathcal{S}$ at iteration t of Algorithm 3.2. This can be achieved in $O(|\mathcal{S}_t|^2)$. However, at each step, the size of $\mathcal{S}_t$ is reduced. Since $\mathcal{S}_0$ is initialized to $\mathcal{D}(\eta)$, the complexity of this operation becomes $O(|\mathcal{D}(\eta)|^2)$, and this is an upper bound on the computational complexity.

To conclude, we need at most $|\mathcal{D}(\eta)|^3$ pairwise comparisons (this bound is not tight), and each pairwise comparison involves a dominance test. Algorithm 3.2 consequently runs in $O(|\mathcal{D}(\eta)|^3 \cdot f(\mathcal{L}(\eta)))$, and this bound is not tight either. □

Implications of Property 3.2

Let $\mathcal{S}$ denote a set of super local upper bounds satisfying Property 3.2.

Proposition 3.2. *Suppose that $|\mathcal{S}| \geq 2$ and let $s^1, s^2 \in \mathcal{S}$, $s^1 \neq s^2$, be two super local upper bounds. For any pair $u^1 \in \mathcal{D}(\eta, s^1)$, $u^2 \in \mathcal{D}(\eta, s^2)$, the intersection point u^I of u^1 and u^2 is not dominated by the lower bound set $\mathcal{L}(\eta)$.*

Proof. The point s^I (intersection point of s^1 and s^2) is not dominated by the lower bound set since $s^1, s^2 \in \mathcal{S}$ and $\mathcal{S}$ satisfies Property 3.2b. Furthermore, by Definition 3.5, $s^1_k \geq u^1_k$ and $s^2_k \geq u^2_k$, $\forall k \in \{1, ..., p\}$. Hence, $u^I_k = \min\{u^1_k, u^2_k\} \leq \min\{s^1_k, s^2_k\} = s^I_k$, $\forall k \in \{1, ..., p\}$. In other words, $u^I \leqq s^I$. Thus, since s^I is not dominated by the lower bound set, u^I is not dominated by the lower bound set either. □

Lemma 3.2. *Let $s \in \mathcal{S}$ be a super local upper bound such that $|\mathcal{D}(\eta, s)| \geq 2$. For any $u \in \mathcal{D}(\eta, s)$, there exists $u' \in \mathcal{D}(\eta, s)$, $u \neq u'$, such that the intersection point of u and u' is dominated by the lower bound set $\mathcal{L}(\eta)$.*

Proof. Suppose that there exists $u \in \mathcal{D}(\eta, s)$ such that there exists no distinct $u' \in \mathcal{D}(\eta, s)$ such that their intersection point is dominated by the lower bound set. Then it is possible to split s into two super local upper bounds, s^1 and s^2, such that $\mathcal{D}(\eta, s^1) = \{u\}$ and $\mathcal{D}(\eta, s^2) = \mathcal{D}(\eta, s) \setminus \{u\}$. The super local upper bounds s^1 and s^2 satisfy Property 3.2a, Property 3.2b and Property 3.2c, and therefore Property 3.2d is not satisfied. □

Lemma 3.3. *If $|\mathcal{S}| \geq 2$, there is no $s^1, s^2 \in \mathcal{S}$, $s^1 \neq s^2$, such that $\mathcal{D}(\eta, s^1) \subseteq \mathcal{D}(\eta, s^2)$.*

Proof. Suppose that there exist $s^1, s^2 \in \mathcal{S}$ such that $\mathcal{D}(\eta, s^1) \subseteq \mathcal{D}(\eta, s^2)$. Let s^I be the intersection point of s^1 and s^2. In particular, by Property 3.2c, the super local upper bounds are as tight as possible, and therefore $s^1 \leqq s^2$. Consequently, $s^1_k \leq s^2_k$, $\forall k \in \{1, ..., p\}$ and thus $s^I = s^1$.

Now, the fact that s^I is dominated by the lower bound set has to be shown. By construction of the super local upper bounds, $s^1 \geqq u$, $\forall u \in \mathcal{D}(\eta, s^1)$. Furthermore, by construction again, each local upper bound in $\mathcal{D}(\eta, s)$ is dominated by the lower bound set. Thus, $s^I = s^1$ is dominated by the lower bound set, and Property 3.2b is not satisfied. □

Lemma 3.4. *Let $\mathcal{S} = \{s^1, ..., s^t\}$ be a set of super local upper bounds. The sets $\mathcal{D}(\eta, s^1), ..., \mathcal{D}(\eta, s^t)$ form a partition of $\mathcal{D}(\eta)$.*

Proof. If $|\mathcal{S}| = 1$, each local upper bound will be included in the unique super local upper bound $s \in \mathcal{S}$. In particular, $\mathcal{D}(\eta, s) = \mathcal{D}(\eta)$ in this case and thus, $\mathcal{S}$ is a partition of $\mathcal{D}(\eta)$.

If $|\mathcal{S}| \geq 2$, and since $\mathcal{S}$ satisfies Property 3.2 and in particular Property 3.2a, it can immediately be concluded that each $u \in \mathcal{D}(\eta)$ is included in at least one $\mathcal{D}(\eta, s^i)$, for $i \in \{1, ..., t\}$.

We now have to prove the following statement: each $u \in \mathcal{D}(\eta)$ is included in at most one set $\mathcal{D}(\eta, s^i)$, for $i \in \{1, ..., t\}$. Suppose that there exists $u \in \mathcal{D}(\eta)$ such that $u \in \mathcal{D}(\eta, s^1)$ and $u \in \mathcal{D}(\eta, s^2)$, $s^1, s^2 \in \mathcal{S}$, $s^1 \neq s^2$.

Lemma 3.2 says that there exists $u' \in \mathcal{D}(\eta, s^1)$ such that the intersection point of u and u' is dominated by the lower bound set, which means that they have to be merged. Similarly, there exists $u'' \in \mathcal{D}(\eta, s^2)$ such that the intersection point of u and u'' is dominated by the lower bound set and they have to be merged. It can be noticed that u' has to be merged with u, that has to be merged with u''. Consequently, u, u' and u'' have to be put in a common super local upper bound.

If $u', u'' \notin \mathcal{D}(\eta, s^1) \cap \mathcal{D}(\eta, s^2)$, the conclusion is that s^1 and s^2 have to be merged, which contradicts Property 3.2b. If this is not the case, the same principle applies to u' and u''. As both Lemma 3.2, Lemma 3.3, and $s^1 \neq s^2$ hold true, the situation where s^1 and s^2 have to be merged will always be reached, and this will contradict Property 3.2b.

□

Theorem 3.3. *The set $\mathcal{S}$ is unique.*

Proof. If $|\mathcal{S}| = 1$, then by Lemma 3.4, $\mathcal{S}$ is unique. Now we study the case where $|\mathcal{S}| \geq 2$.

Let $\mathcal{S}$ and $\mathcal{S}'$ be two sets of super local upper bounds satisfying Property 3.2 and such that $\mathcal{S} \neq \mathcal{S}'$. Hence, there exist at least two sets $\mathcal{D}(\eta, s)$ and $\mathcal{D}(\eta, s')$, respectively, from $\mathcal{S}$ and $\mathcal{S}'$ that are different. Furthermore, it is always possible to find $\mathcal{D}(\eta, s)$ and $\mathcal{D}(\eta, s')$ such that they have at least one common element. Otherwise, by re-indexing the sets, the same partition would be obtained, leading to $\mathcal{S} = \mathcal{S}'$, which is a contradiction.

Note that since $\mathcal{S}$ and $\mathcal{S}'$ are different, then necessarily $|\mathcal{D}(\eta, s)| \geq 2$ and $|\mathcal{D}(\eta, s')| \geq 2$. Otherwise, because of this common element u, we would have $\mathcal{D}(\eta, s) \subset \mathcal{D}(\eta, s')$ or $\mathcal{D}(\eta, s') \subset \mathcal{D}(\eta, s)$, which is not possible because of Lemma 3.2. Indeed, if $\mathcal{D}(\eta, s) \subset \mathcal{D}(\eta, s')$, then there exists $v \in \mathcal{D}(\eta, s')$ such that $v \notin \mathcal{D}(\eta, s)$, and it has an intersection point with another local upper bound in $\mathcal{D}(\eta, s)$ that is dominated by the lower bound set (Lemma 3.2), which leads to the conclusion that s has to be merged with another super local upper bound in $\mathcal{S}$ (the one that contains v), which contradicts the fact that $\mathcal{S}$ satisfies Property 3.2b. Equivalently, the same reasoning can be applied to the case where $\mathcal{D}(\eta, s') \subset \mathcal{D}(\eta, s)$.

This means that there exists $u \in \mathcal{D}(\eta)$ such that $u \in \mathcal{D}(\eta, s)$ and $u \in \mathcal{D}(\eta, s')$ with $\mathcal{D}(\eta, s) \neq \mathcal{D}(\eta, s')$ and such that:

- $\exists v \in \mathcal{D}(\eta, s)$ such that $v \notin \mathcal{D}(\eta, s')$ and the intersection point of u and v is dominated by the lower bound set (because of Lemma 3.2);
- $\exists v' \in \mathcal{D}(\eta, s')$ such that $v' \notin \mathcal{D}(\eta, s)$ and the intersection point of u and v' is dominated by the lower bound set (because of Lemma 3.2).

By definition, v and u then have to be merged in the same super local upper bound. Since $v \notin \mathcal{D}(\eta, s')$ and $\mathcal{S}'$ is a partition of $\mathcal{D}(\eta)$ (Lemma 3.4), then there exists $\mathcal{D}(\eta, \hat{s}')$ such that $\hat{s}' \in \mathcal{S}'$ and $v \in \mathcal{D}(\eta, \hat{s}')$. By construction, $\mathcal{D}(\eta, s')$ and $\mathcal{D}(\eta, \hat{s}')$ have to be merged, and this contradicts the fact that $\mathcal{S}'$ satisfies Property 3.2b. The same reasoning can be applied to v' and $\mathcal{S}$. □

3.4.3 An alternative branching strategy using an upper bound on the objectives

In the previous section, inequalities were derived from the partial dominance of the lower bound set by the upper bound set and used to create additional subproblems in the objective space. We aimed at creating a maximum number of subproblems in order to discard as much dominated area as possible, thereby satisfying Property 3.1c. More subproblems may obviously lead to more nodes in the branching tree, and from a practical point of view this may lead to prolonged computation times if the nodes are not explored sufficiently fast. In this case, a new question arises: is it possible to derive inequalities from the partial dominance of the lower bound set without generating a large number of subproblems?

This can be achieved by modifying Algorithm 3.2. At each node η, by choosing to always merge the dominated local upper bounds instead of only merging when their intersection point is dominated by the lower bound set, a unique super local upper bound s is always obtained. This super local upper bound actually corresponds to the nadir point $d^N(\eta)$ of all the super local upper bounds $\mathcal{D}(\eta)$. Hence Step 4 of Algorithm 3.1 becomes *add constraints* $Cx \leqq d^N(\eta)$ *to the sub-problem*, and a single unique child node is created in the branching tree. Next, two child nodes are created due to variable splitting in the decision space (Step 5). That is, we obtain two disjoint sub-problems with an upper bound on the objectives.

3.5 Computational experiments

In this section, we report the results of the computational experiments conducted with the multi-objective Branch-and-Bound (B&B) algorithm. All algorithms were implemented in C++17. The experiments were carried out on a computer with an Intel(R) Core(TM) i7-4800MQ CPU @ 2.70GHz processor and 32GM of RAM memory, on Windows 10 with a time a limit of one hour (3600 seconds). The implementation is available at Forget (2021). No parallelization was used in the branch-and-bound algorithm itself, and Cplex's default parameters were used.

When selecting a node to explore (Step 1 of Algorithm 3.1), a breadth-first strategy is adopted. Preliminary experiments showed that there was no clear winner between breadth-first and depth-first strategies for the problem classes that we considered. Breadth-first was chosen as it had the best average performance over all instances.

The algorithm from Forget et al. (2022) was used for the computation of the linear relaxation at each node (Step 2 of Algorithm 3.1). In particular, this algorithm is based on Benson's outer approximation algorithm (see Benson (1998), and further improvements in Hamel et al. (2013), Csirmaz (2015), Löhne and Weißing (2020)). Forget et al. (2022) accelerate the solution process in the specific context of the B&B by warmstarting the algorithm using the lower bound set from the father node. Only in the root node, the linear relaxation was computed from scratch as no father node is available. All single-objective linear programs are solved using CPLEX 12.10.

Preliminary tests were performed to understand if the full lower bound set (i.e. the relaxation) should be computed at each node. The tests revealed that many potentially non-dominated points are gathered from the extreme points of the lower bound set. Computations that did not completely compute the linear relaxation led to many vertices for which the pre-image was missing and thus, to upper bound sets of much worse quality. Ultimately, this resulted in worse performances with respect to CPU time.

The branching variable selected in Step 5 of Algorithm 3.1 differs depending on whether objective branching is applied or not. If no objective branching is performed, the algorithm will branch on the free variable that is the most often fractional among the extreme points of the lower bound set, given that at least one of the variables takes a fractional value. If no variable takes a fractional value in any of the extreme points, the variable that differs in value most often (i.e., with the average value closest to 0.5) is chosen. If objective branching is enabled, the rule is the same, except that a different variable may be chosen in each subproblem. In the case where objective branching is applied on $s \in \mathbb{R}^p$, only the extreme points of the lower bound set included in $\mathcal{C}(s)$ will be considered. If multiple choices are possible or if no extreme point is located in $\mathcal{C}(s)$, the free variable with the smallest index is chosen.

To test different algorithm configurations, three objective-space-related rules are considered:

- `noOB`: no objective branching is performed. This is equivalent to skipping Step 4 of Algorithm 3.1;
- `fullOB`: as many sub-problems as possible are created in the objective space, but no redundancies are allowed. This is full objective branching as described using super local upper bounds in Algorithm 3.2;
- `coneB`: no branching is performed in the objective space, but an upper bound on the objectives is derived from the dominance test. The upper bound is the nadir point of the local upper bounds dominated by the lower bound set (see Section 3.4.3). This is referred as cone bounding. A single node is created in the branching tree.

The purpose of the computational study is to answer the following questions:

- How do the different algorithm configurations perform, and which configurations perform the best? In particular, is objective branching worthwhile (Section 3.5.2)?
- Why does objective branching perform the way it does (Section 3.5.3)? This includes an analysis of how an increasing number of objectives affect objective branching.
- What does the structure of the search tree look like when full objective branching is used (Section 3.5.4)?
- How does the B&B algorithm perform compared to an objective space search algorithm (Section 3.5.5)?

We emphasize that the purpose of this study is to lay the ground for efficient and strong bounding strategies in multi-objective branch and bound algorithms and hence to initialize a new line of research in this direction. As a consequence, the focus of our work and of the computational study is on bound computations rather than on the generation of cutting planes and efficient preprocessing strategies in the overall branch and bound framework. Nevertheless, we report comparisons with state-of-the-art objective space search algorithms. These comparisons have to be carefully evaluated. Indeed, objective space search methods benefit from the great efficiency of MIP solvers like, for example, CPLEX, that rely on extensive and long-standing algorithmic developments. Depending on the considered problem class, this can be expected to outplay the potential advantage of multi-objective branch and bound methods that perform the search in the decision space and thus avoid the repeated solution of independent ε-constraint IPs.

3.5.1 Test instances

A total of 600 instances (see Table 3.1) taken from the multi-objective literature have been used. Four problem classes are considered: Assignment Problems (AP) from Bektaş (2018), randomly generated Integer Linear Programs (ILP) and Knapsack Problems (KP) from Kirlik (2014) (online at Forget, Nielsen, and Gadegaard (2020d)), Uncapacitated Facility Location Problems (UFLP), and Production Planning Problems (PPP) (online at Forget, Nielsen, and Gadegaard (2020c)). A total of 10 instances are solved for each number of objectives and number of variables. The number of variables in each problem class was increased until none of the algorithm configurations were able to compute the non-dominated set within a time limit of 3600 seconds for several instances. Instances with 3, 4 and 5 objective functions are considered.

All instances are converted to minimization problems, meaning that if an objective function $z(x)$ should be maximized, $-z(x)$ is minimized instead. Furthermore, all instances have integer coefficients only. Hence, integer rounding was used in the dominance test, where local upper bounds were shifted by -1 on each objective; and in the objective branching

Table 3.1: Instances used (600 instances in total).

Class	p[a]	n[b]	#[c]
AP	3	100, 225, 400, 625, 900	50
AP	4	25, 100, 144, 225	40
AP	5	25, 36, 49, 64	40
ILP	3	10, 20, 30, 40	40
ILP	4	10, 20, 30	30
ILP	5	10, 20	20
KP	3	10, 20, 30, 40, 50	50
KP	4	10, 20, 30, 40	40
KP	5	10, 20	20
PPP	3	33, 45, 54, 63, 72	50
PPP	4	24, 27, 33, 39, 48, 57	60
PPP	5	15, 18, 24, 36	40
UFLP	3	42, 56, 72, 90	40
UFLP	4	20, 30, 42, 56	40
UFLP	5	12, 20, 30, 42	40

[a] Number of objectives.

[b] Variable sizes.

[c] Number of instances.

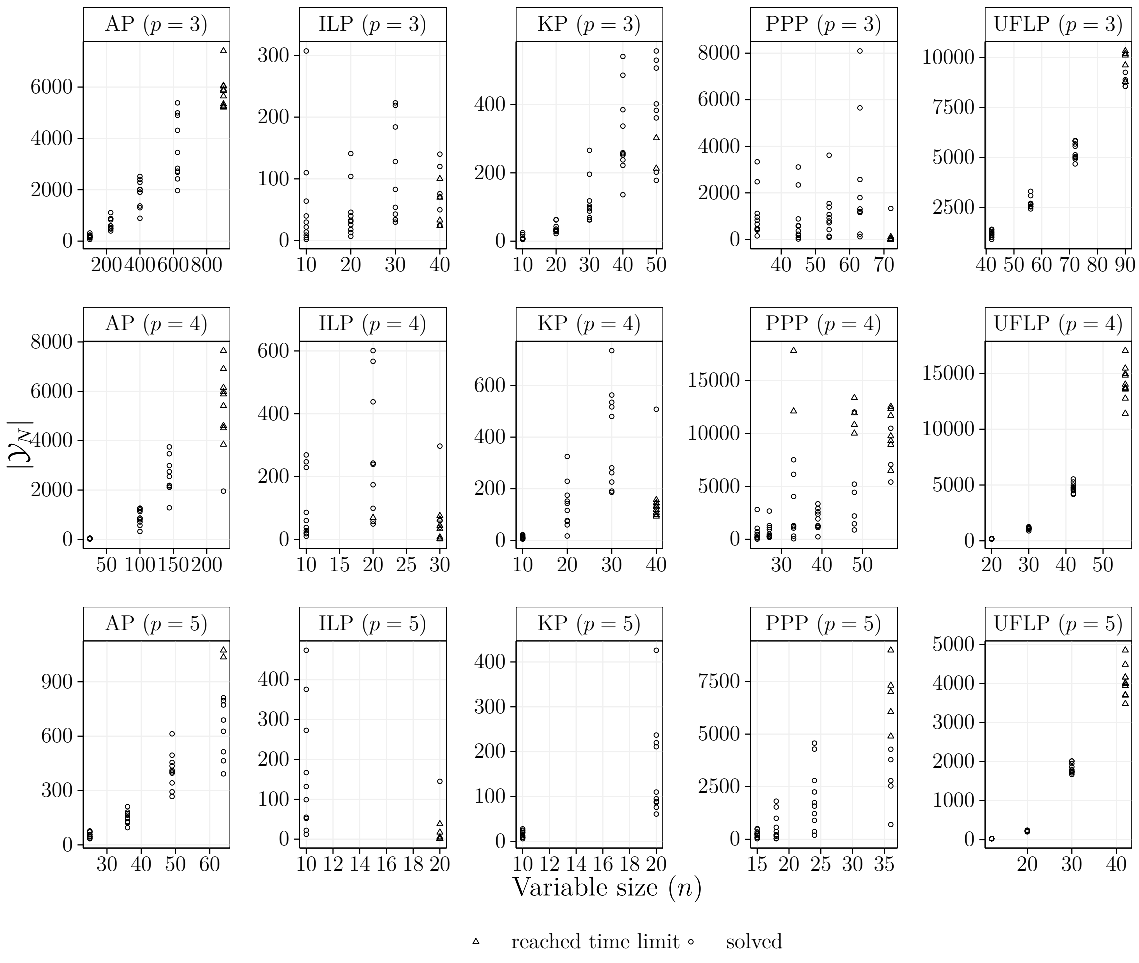


Figure 3.5: Number of non-dominated points. One point for each instance is given. Instances that have not been solved to optimality are illustrated with a different shape. Note that the scale for each sub-plot is different.

constraints when computing the linear relaxation, where $Cx \leqq s - 1$ was used instead of $Cx \leqq s$ when objective branching was applied on the super local upper bound $s \in \mathbb{R}^p$. Note that all configurations were tested both with and without integer rounding. For ILP, KP, UFLP, and PPP, the benefit of integer rounding was very low (between 0 and 3% of speed-up), whereas it had a larger impact for AP (22% of speed-up on average). This seems to be correlated with the ranges of the coefficients of the objective functions. Indeed, the coefficients are in the interval $[1, 20]$ for AP whereas they are in the range $[1, 100]$ or $[1, 1000]$ for the other problem classes. Moreover, there was no correlation between the percentage of speed-up and the configuration used for the branch-and-bound.

In Figure 3.5 the number of non-dominated points are given for each instance. We have increased the variable size for each problem class until the size becomes so large that some or all instances cannot be solved within the time limit. The instances which have not been solved to optimality (18%) are illustrated with a different shape. In general, the number of non-dominated points grows with variable size (n) and number of objectives (p). Note though that there may be a high variation for fixed n and p. Moreover, the variation grows with n and p. For UFLP, the number of non-dominated points grows rapidly as a function of variable size which is due to the high percentage of objective coefficients not dominated by other coefficients.

3.5.2 Performance of the different algorithm configurations

A comparison of the different algorithm configurations is given in Figure 3.6 where the ratio with `noOB` as benchmark is plotted. We limit the analysis to the set of instances that were solved to optimality for all algorithm configurations (75% of the instances). First, observe that the performance of objective branching (`coneB` and `fullOB`) is problem dependent. Full objective branching and cone bounding perform well for AP, ILP, KP, okay for PPP and poorly for UFLP. Moreover, the variation in performance is higher for PPP and UFLP. Second, the performance is highly affected by the number of objectives. For $p = 3$ the CPU time of `coneB` and `fullOB` decreases with 22% and 9% compared to `noOB`, respectively. But the performance deteriorates as p increases: for $p = 5$ the decrease using `coneB` and `fullOB` in the CPU compared to `noOB` is -4% and -10%, respectively. That is, `noOB` performs overall best for $p = 5$. Finally, note that in general, `fullOB` and `coneB` perform very similarly. The exception is for UFLP (and partly PPP), where `coneB` performs systematically better than `fullOB`.

Possible reasons for these observations will be elaborated upon in the next sections.

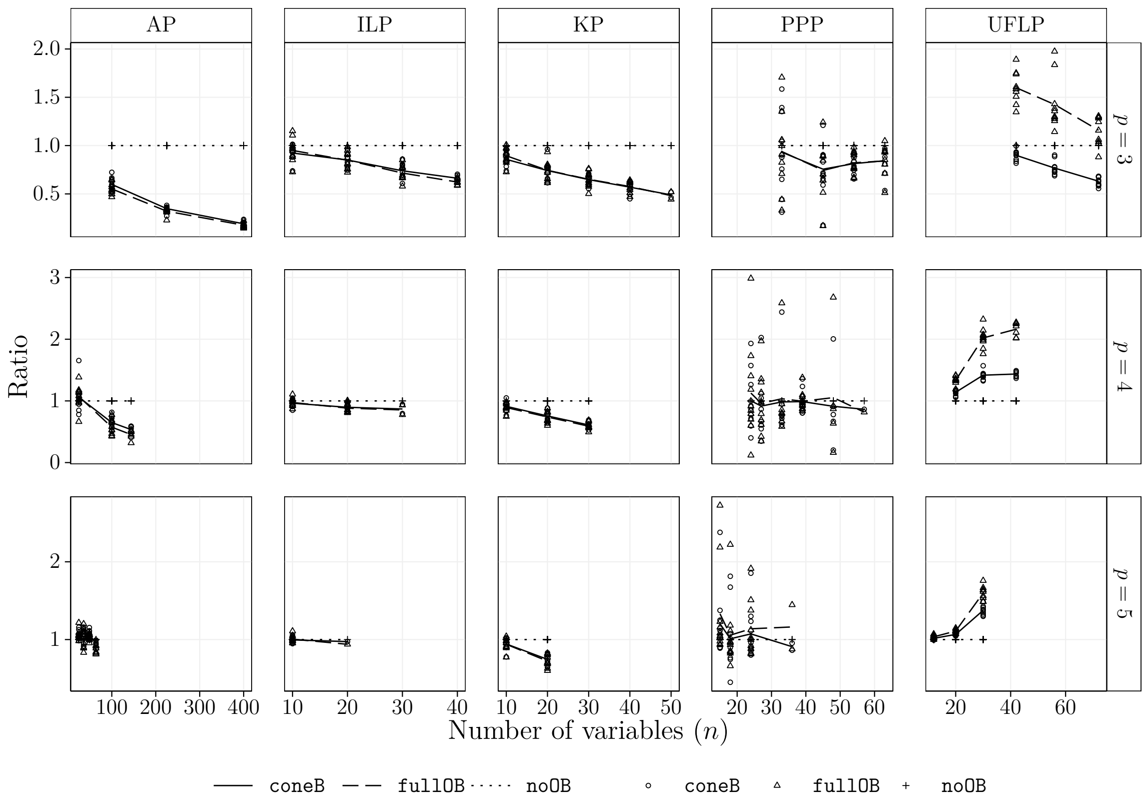


Figure 3.6: CPU time ratio (CPU time divided with the CPU using `noOB`) for each test instance (points) together with averages (lines).

3.5.3 Objective branching: a closer look

To take a closer look at the different objective branching configurations, we limit the analysis to the set of instances that were solved to optimality for all algorithm configurations (75% of the instances). In this section, we aim at understanding the reasons why objective branching performs the way it does.

The node ratio of the branching tree is depicted in Figure 3.7. First, observe that using `coneB` systematically leads to smaller (or similar) trees compared to `noOB`. The only impact `coneB` has on the sequence of branching decisions compared to `noOB`, is on which extreme points are considered when deciding on the next branching variable. This highlights the fact that choosing the next branching variable based on parts of the objective space where LB set is not dominated by the UB set is a good strategy for obtaining smaller trees.

Second, observe that `fullOB` often leads to larger (or similar) trees than `noOB` for ILP, KP, and PPP, but instances are solved faster with `fullOB`. This implies that there are other benefits to objective branching than just smarter branching decisions when using `coneB`. Two reasons may be pointed out:

- When using objective branching, we restrict the computation of the lower bound set

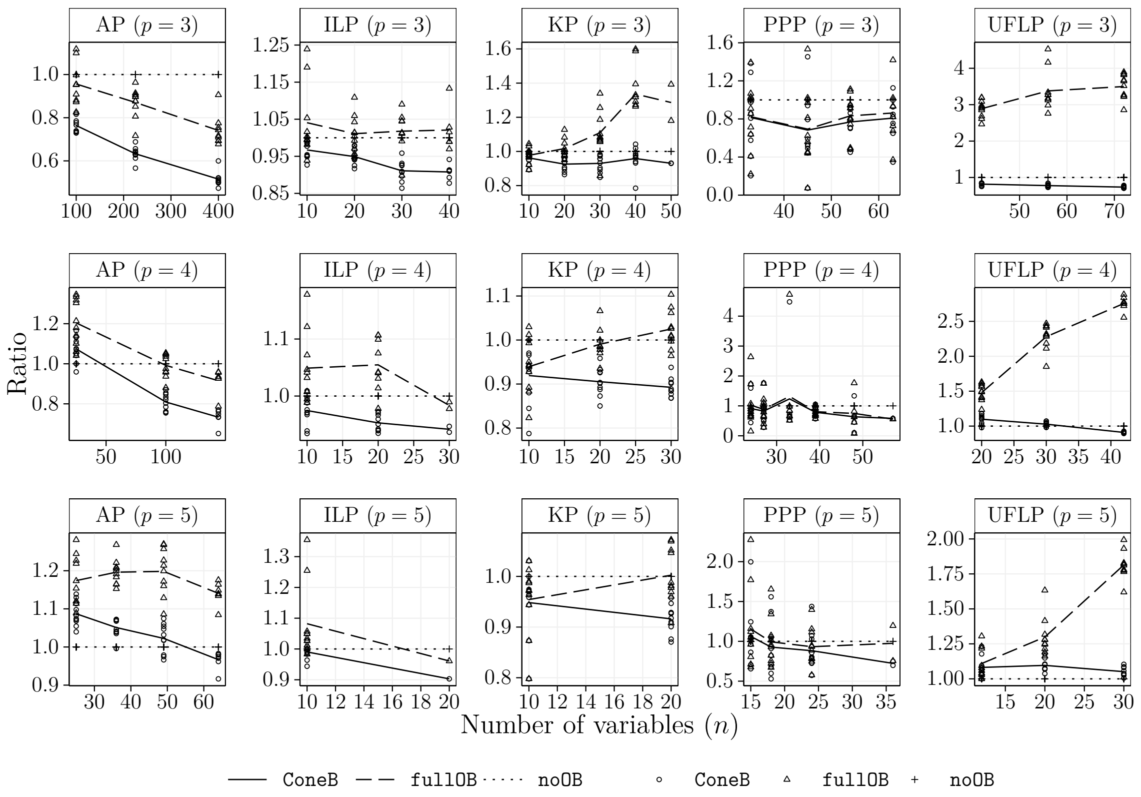


Figure 3.7: Nodes in the branching tree ratio (number of nodes divided with the number of nodes using `noOB`) for each test instance (points) together with averages (lines).

to a specific part of the objective space. Hence, contrary to `noOB`, the computation of lower bound sets is avoided in areas of the objective space already known as dominated. This leads to smaller lower bound sets, which is beneficial since computing the linear relaxation is the most time consuming part (see Forget et al. (2022)).

- The way nodes are fathomed in the tree is significantly impacted by objective branching (either `coneB` or `fullOB`). Figure 3.8 shows how the nodes are fathomed in proportion to the total number of leaf nodes. Note that the proportion of nodes fathomed by infeasibility tends to become much larger when objective branching is used. Fathoming a node by infeasibility is the fastest way to fathom a node since it occurs in the start when processing a node while fathoming by dominance or optimality requires to have the lower bound set computed. Hence, leaf nodes can be fathomed faster when using objective branching.

Although these two reasons may improve performance, there are other mechanisms that may have a negative impact on performance when using objective branching:

- As presented in Section 4, when using objective branching, it is necessary to check for the dominance status of every single local upper bound during the dominance test,

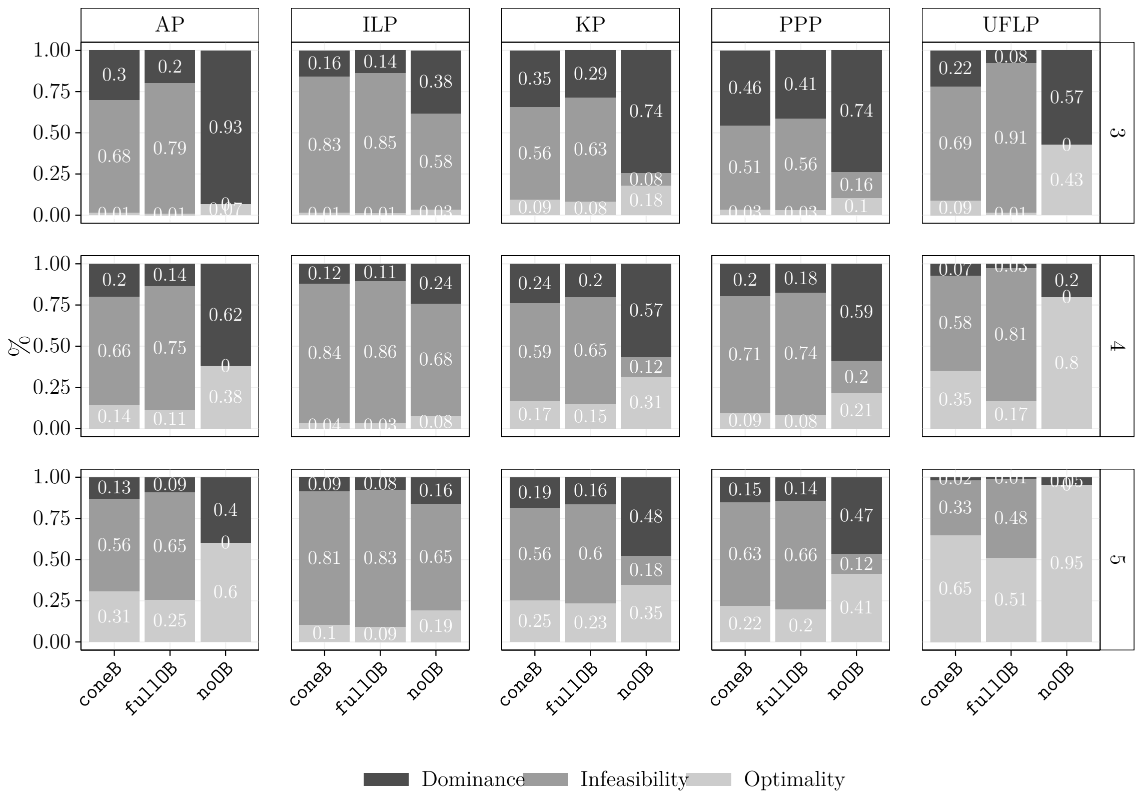


Figure 3.8: Proportion of leaf nodes fathomed by dominance, infeasibility, and optimality.

while it can be stopped when a dominated one is found with `noOB`.

- Due to Theorem 3.2, the cost of computing the super local upper bounds when using `fullOB` is $O(|\mathcal{D}(\eta)|^3)$ where $\mathcal{D}(\eta)$ denote the local upper bounds. That is, the CPU time may increase if the number of local upper bounds increases. For $p = 3$, the number of local upper bounds per non-dominated point is bounded (Dächert and Klamroth, 2015). However, no bound is known when $p \geq 4$. Even worse, the computational study of Klamroth et al. (2015) showed that the number of local upper bounds seems to grow exponentially as a function of the number of objectives for a given number of non-dominated points (approx. 7 local upper bounds per non-dominated point for $p = 4$ and 32 when $p = 5$).
- Many super local upper bounds may result in a too large number of child nodes when applying `fullOB`. For example, `fullOB` tends to develop significantly larger trees than `noOB` for UFLP, and it appears that the previously enumerated benefits are not enough to compensate for the higher number of nodes created.

Hence, a high number of non-dominated points may result in a high number of local upper bounds that are costly to check for dominance and may result in too many child nodes.

	p[a]	% of nodes[b]	Min depth[c]	Avg-. # nodes[d]	Max # nodes[e]
AP	$p = 3$	5.01	3.64	2.66	27
	$p = 4$	4.08	4.8	2.41	16
	$p = 5$	3.38	5.3	2.25	8
ILP	$p = 3$	2.52	10.41	2.17	11
	$p = 4$	2.42	9.96	2.27	11
	$p = 5$	2.09	9.31	2.19	14
KP	$p = 3$	4.21	6.52	2.26	16
	$p = 4$	3.09	6.66	2.24	12
	$p = 5$	1.76	6.20	2.14	7
PPP	$p = 3$	2.25	12.43	2.27	32
	$p = 4$	2.39	9.67	2.25	50
	$p = 5$	2.15	6.85	2.33	39
UFLP	$p = 3$	5.44	3.59	4.51	157
	$p = 4$	4.85	5.25	3.52	144
	$p = 5$	2.58	6.59	2.97	51

[a] Number of objectives.
[b] Percentage of nodes where objective branching resulted in two or more child nodes.
[c] Average minimum depth at which objective branching resulted in two or more child nodes.
[d] Average number of child nodes created when objective branching resulted in two or more child nodes.
[e] Maximum number of child nodes created when objective branching resulted in two or more child nodes.

Table 3.2: Average percentage of nodes, average minimal depth and average and maximum number of child nodes created when at least two or more child nodes are created due to objective branching.

Moreover, these negative effects on CPU increase with increasing number of objectives as can be seen in Figure 3.6 for PPP and UFLP which indeed are instances with a high number of non-dominated points.

3.5.4 Branching tree structure when using `fullOB`

In this section, we investigate the structure of the tree when `fullOB` is used and we restrict the analysis to instances for which `fullOB` was solved to optimality. Branching tree statistics are given in Table 3.2.

First, observe that on average `fullOB` separates the problem into two or more disjoint sub-problems in a very small proportion of the nodes (between 1.8% and 5.4% of the nodes on average). This suggests that it is often not possible to perform disjoint separation of the objective space using objective branching. Moreover, given a problem class this proportion tends to decrease as p increases. This suggest that the difficulty of applying objective

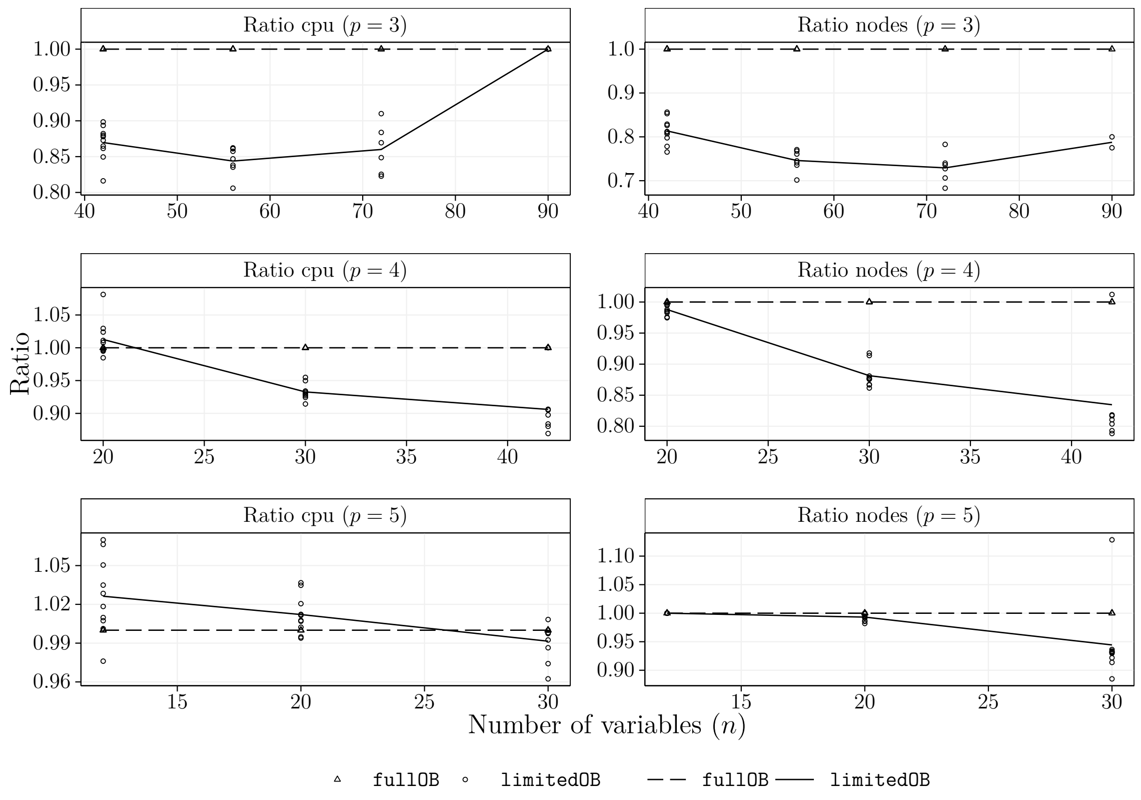


Figure 3.9: Ratios of comparing CPU and nodes in the tree using `fullOB` as benchmark (in the denominator when dividing the numbers) for UFLP instances.

branching keeps increasing with the number of dimensions.

Second, an interesting fact is that, unlike in the bi-objective case, objective branching cannot be applied early in the tree (Parragh and Tricoire, 2019). In general, it requires a higher depth in the tree before objective branching can be applied. Moreover, this result holds even though preliminary experiments showed that for some problem classes (AP, PPP, and UFLP), non-dominated points were found very early in the tree, and even at the root node. This supports the observation of the difficulties of applying objective branching with an increasing number of objective functions as presented in Section 3.4.1.

Finally, consider the average and maximum number of child nodes created when applying objective branching. Observe that the average number is very close to the minimum number of nodes created when applying objective branching (two nodes). That is, often only a few nodes are created when applying objective branching. However, in a few cases a larger number of child nodes are created (up to 157 for UFLP). These cases may result in a wide and big tree. Indeed, a possible reason for `fullOB` to perform so poorly for UFLP may be that a large number of subproblems are created early in the tree, resulting in larger sub-trees.

To test how a large number of child nodes affect the branching tree, a new configuration denoted `limitedOB` is considered. Here, an upper limit of 5 is used on the number of child nodes created when applying objective branching. The child nodes are created using Algorithm 3.2, which upon termination, merge the super local upper bounds until the number is at most five. The merging operation merges the two closest super local upper bounds, merges all the super local upper bounds with an intersection point dominated by the lower bound set, and repeats the process until at most five super local upper bounds remain.

The performance of `limitedOB` is shown in Figure 3.9, where ratios are obtained by dividing the numbers for `limitedOB` with `fullOB`. Note that using `limitedOB` reduces the tree size with up to 32% and in general performs better than `fullOB`. This indicates that having an upper bound on the number of child nodes created may help reducing the tree size and improve performance for instances where a high number of child nodes are created.

Any separation in the objective space obtained with Algorithm 3.2 at a given node is also valid for any of its child nodes. That is, both `fullOB` and `limitedOB` separate the objective space. However, due to the upper limit on the number of child nodes, the separation for `limitedOB` is not as tight as for `fullOB`. Moreover the separations applied when using `fullOB` remain valid and may be applied later in the sub-tree when using `limitedOB`, i.e. `limitedOB` may “delay” objective branching to deeper levels of the tree by applying the separations in smaller steps.

3.5.5 Comparison with an Objective Space Search algorithm

We now compare the performance of the branch-and-bound algorithm using objective branching to several Objective Space Search algorithms (OSS). In doing so, we emphasize that this comparison serves as a proof of concept rather than as a validation of the superiority of our approach. Indeed, OSS methods are based on the iterative solution of single-objective IPs, for which excellent solvers are available. Even though our branch-and-bound implementation avoids the repeated consideration of the same - or very similar - (partial) solutions in the decision space, implementing state-of-the art preprocessing and cut-generation strategies as used within standard IP-solvers was beyond the scope of this work, so that OSS methods have a large advantage in this regard.

We base our comparison on two exemplary OSS methods: The C++ implementation of Kirlik and Sayın (2014), available at Kirlik (2014), and denoted by configuration `OSS-KS`. In addition, a C++ implementation of the redundancy avoidance method introduced in Klamroth et al. (2015) and implemented in Dächert, Fleuren, and Klamroth (2021) is used for comparison, and denoted by configuration `OSS-DFK`. The authors are aware of other and more recent OSS algorithms, such as Bektaş (2018); Holzmann and Smith (2018); Tamby and Vanderpooten (2021). The above methods were selected for two reasons: `OSS-KS` is used in almost all comparative studies involving OSS methods and can thus be seen as a general reference. `OSS-DFK` implements the idea of redundancy avoidance while keeping the IPs

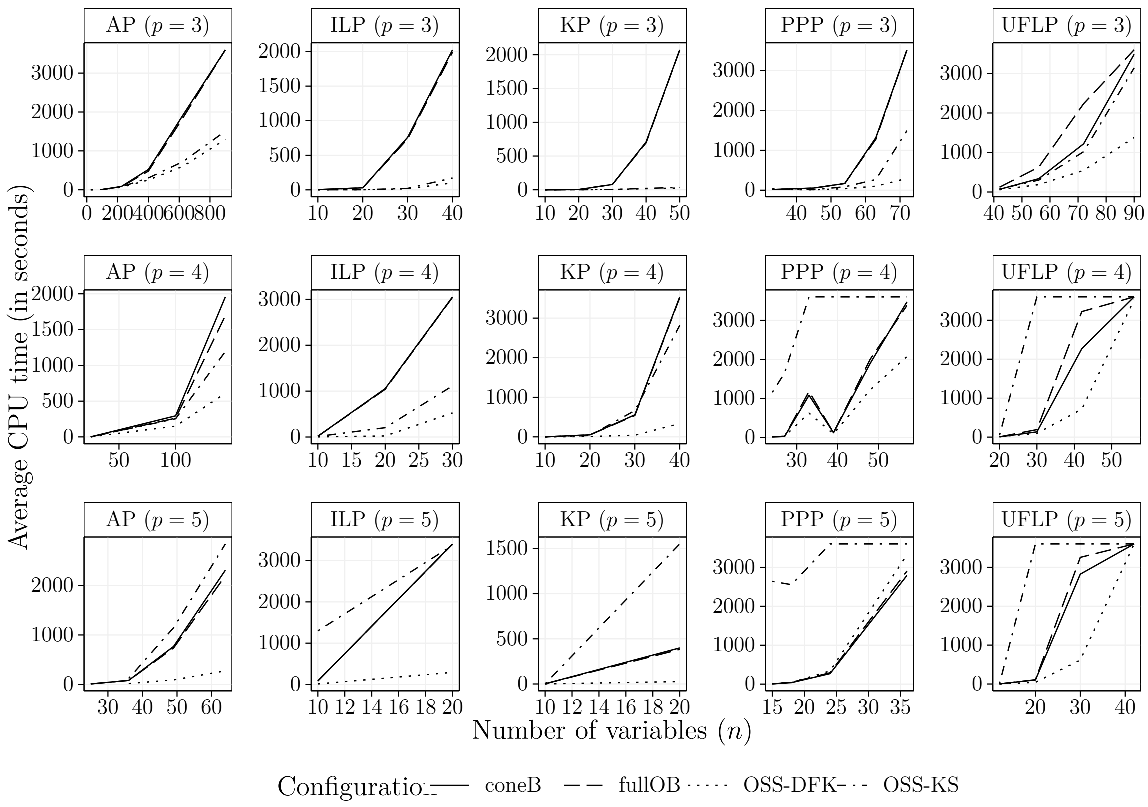


Figure 3.10: Average cpu times, expressed in seconds, in function of the number of variables for each number of objectives and problem classes. Unsolved instances are included here. Four configurations are depicted: `fullOB`, `coneB`, `OSS-DFK`, and `OSS-KS`.

simple and can thus be seen as a good compromise between IP complexity and the number of required solver-calls. Tamby and Vanderpooten (2021) present an improved implementation of the same method that uses additional, partially problem and solver specific features like providing starting solutions to `CPLEX` and re-ordering the subproblems in a clever way. The improvements are thus rather in the details than in the overall concept. Holzmann and Smith (2018) use weighted Tchebychev scalarizations rather than e-constraint scalarizations and present promising results, particularly when using again a re-implementation of Klamroth et al. (2015), i.e., the same method used here for comparison. Bektaş (2018) reduces the number of IPs, however, at the price of solving more complicated IPs that involve disjunctive programming formulations. Their reported improvements over Kirlik and Sayın (2014) are, however, lower than those reported in Tamby and Vanderpooten (2021), a method that we mimic here.

The results are given in Figure 3.10. In general, the OSS algorithm performs better than the B&B algorithm for $p = 3$. For ILP, KP and PPP, the gap is significant. For UFLP, although OSS is still better on average, the gap is smaller. For $p = 4$, `OSS-DFK` is the best configuration. For PPP, a smaller gap compared to $p = 3$ is observed. This is also observed for $p = 5$, where in fact the branch-and-bound method appears to be slightly faster than `OSS-DFK`.

It does not come as a surprise that the OSS algorithm is highly competitive and outperforms current state-of-art multi-objective B&B algorithms in most cases. It benefits from the power of single-objective MIP solvers, which have improved over decades. Having this in mind, the purpose of this study is not necessarily to outperform the OSS algorithm, but rather to discuss and thoroughly analyze the concept of objective branching in a B&B algorithm (that has proven successful for bi-objective problems) in the multi-objective case. Our aim is that these efforts will result in an advancement of the development of promising methods that hybridize decision space and objective space search methods.

For increasing number of objectives and number of non-dominated points the B&B algorithm becomes more competitive. Indeed, UFLP is the problem class with the largest size of the non-dominated set (2298 on average for $p = 3$ over all instances solved by the branch-and-bound), and this is where the gap is the smallest. Moreover, PPP records one of the largest number of non-dominated points (1290 on average) and B&B outperform the OSS algorithms for $p = 5$. This is also visible for AP with $p = 3$, for which a large number of non-dominated points as well as a smaller gap is recorded. As a comparison, ILP and KP have 67 and 148 non-dominated points on average, respectively. This suggests that the branch-and-bound is more likely to be efficient on instances with a large number of non-dominated points and objectives.

3.6 Conclusion

This paper proposes an extension of objective branching, a successful feature of bi-objective B&B, to the case with more than two objectives ($p \geq 3$), and studies its impact on a B&B algorithm for multi-objective linear integer programming problems.

First, we highlighted a set of difficulties when extending objective branching from the bi-objective case to the case with more than two objectives.

Second, in order to overcome these difficulties, a number of desirable properties of objective branching was exposed, and the concept of super local upper bounds was introduced. The super local upper bounds were built by merging local upper bounds and were used to define subproblems satisfying the desirable properties previously established.

Next, the experiments in Section 3.5 showed that in general, except for UFLP, either full objective branching or cone bounding performed better, or at least as well as the reference framework without objective branching. In these cases, full objective branching and cone bounding resulted in similar cpu times. The largest benefits were recorded for $p = 3$.

Finally, the experiments showed that there was a positive impact on the variable selection when objective branching or cone bounding was applied, i.e. the algorithm tends to make better branching decisions. Indeed, solutions that are in the same part of the objective space are more likely to be similar, and some variables may not be able to take particular values.

Directions for future research include identifying a reduced set of candidates for variable branching, and to understand what constitutes a good variable to branch on. Moreover, preliminary tests showed that there is no clear winner between depth and breadth first strategies when selecting a node, and the difference can be significant. We believe that it would be beneficial for multi-objective branch-and-bound frameworks to be able to either detect which rule is the best given the instance provided (without knowing the problem class), or to design an alternative rule that works efficiently for all problem classes. Objective branching could also benefit from parallelization since sub-problems are made in a way such that they are independent. Indeed, when objective branching is applied, there is no point of the search area that is included in more than one sub-problem and the sub-problems are defined by different points of the upper bound set. Hence, in each sub-problem, the upper bound set can be improved in the region defined by the objective branching constraints, without influencing the other sub-problems. At last, it is well-known that OSS algorithms benefit from the power of single-objective MIP solvers. A potential line of future research, inspired by what is done in the single-objective case, is to improve the branch-and-bound framework by, for example, exploring cutting planes.

Fourth Chapter

Enhancing branch-and-bound for multi-objective 0-1 programming

History: This chapter has been prepared in collaboration with Sophie Parragh, and initiated during my stay abroad in Linz during the fall semester of 2021. Preliminary results were presented at an internal seminar in Linz in November 2021. After some improvements, the work was presented again at an internal seminar in April 2022 at Aarhus University, Denmark; and a second time at the 26th International Conference on Multiple Criteria Decision Making in June 2022, in Portsmouth, United Kingdom.

Enhancing branch-and-bound for multi-objective 0-1 programming

Nicolas Forget*, Sophie Parragh**

* Department of Economics and Business Economics, School of Business and Social Sciences, Aarhus University, Denmark

** Institute of Production and Logistics Management, Johannes Kepler University, Linz, Austria

Abstract

In bi-objective branch-and-bound literature, a classic way to speed-up the resolution is to perform objective branching, i.e. to create smaller and disjoint sub-problems in the objective space, obtained from the partial dominance of the lower bound set by the upper bound set. When considering three or more objective functions, however, applying objective branching becomes more complex, and the benefit gets unclear. Hence, we investigate whether the objective branching constraints can be further exploited. To do so, we extend the idea of probing to multiple objectives, enhance it in several ways, and show that when coupled with objective branching, it results in significant speed-ups in terms of CPU times. We also investigate cut generation based on the objective branching constraints. Besides, we propose simple node selection rules based on the best-bound principle derived from single-objective literature. We show that the proposed rules outperform the traditional depth-first and breadth-first strategies. All experiments are carried out on a set of both Capacitated and Uncapacitated Facility Location Problems, and on the Knapsack Problem, with three and four objectives.

Keywords: multi-objective combinatorial optimization; multi-objective integer programming; branch & bound; probing; node selection.

4.1 Introduction

In many real-world problem situations, decision makers have to consider several different objectives simultaneously, such as travel times, costs, and CO2 emissions. These objectives are often conflicting, which means that the optimal solution for one of the objectives is often not optimal for the others. Instead, one may search for all the optimal trade-off solutions. For this purpose, a multi-objective optimization problem is solved. We focus here on solving *Multi-Objective Integer Linear Problems* (MOILP).

A MOILP can be solved using an *Objective Space Search* (OSS) algorithm, which consists of solving a series of single-objective problems obtained by *scalarizing* the objective functions so that all optimal trade-offs are enumerated (Ehrgott, 2005). The main advantage of this methodology is that the power of single-objective solvers can be used. Consequently, objective space search algorithms have received much attention over the past decades (see e.g., Ulungu and Teghem (1995); Sylva and Crema (2004); Ozlen et al. (2014); Kirlik and Sayın (2014); Boland et al. (2017); Boland and Savelsbergh (2016); Tamby and Vanderpooten (2021)).

Alternatively, a MOILP can be solved using a *Decision Space Search* (DSS) algorithm, typically a *Multi-Objective Branch & Bound* (MOBB) algorithm. Almost all recent contributions in this area address the bi-objective case (Stidsen et al., 2014; Gadegaard et al., 2019; Parragh and Tricoire, 2019; Adelgren and Gupte, 2022), and they all rely on an efficient branching scheme that creates sub-problems in the objective space. Forget, Gadegaard, Klamroth, Nielsen, and Przybylski (2022) have recently generalized this scheme to problems with more than two objectives. However, in its straightforward form, the speed-ups observed for the two-objective case did not translate to the three or more objective case. In this paper, we take Forget et al. (2022)'s work as the starting point and of we propose an enhanced MOBB framework designed to solve MOILP with three or more objective functions. In order to improve the performance of the MOBB, we generalize the idea of probing to the multi-objective case, which allows us better exploit the constraints generated by the branching scheme. Probing is a technique successfully employed in single objective branch-and-bound, which allows reducing the domains on the decision variables. In a 0-1 integer context, this results in fixing variables to either 0 or 1 (Savelsbergh, 1994). To the best of our knowledge, probing has not been generalized to more than two objectives. Moreover, we investigate whether a decrease in CPU times can be achieved by deriving stronger cuts from the constraints generated by the objective branching scheme. Then, we investigate new node selection rules based on the best-bound principle, and compare them to the traditional breadth and depth-first strategies typically used in MOBB literature. Finally, we show through a computational study that the suggested improvements led to a significant speed-up for the framework in terms of CPU time, and that our algorithm is competitive with recent OSS algorithms from the literature on some of the problem classes considered in this paper.

The paper is organized as follows. In Section 4.2, we present notations and definitions used throughout the paper. In Section 4.3, we discuss related work, and in Section 4.4, we describe the basic MOBB framework used here. Sections 4.5 and 4.6 are dedicated to the main novelties of our framework, namely probing, cut generation, and node selection rules. Finally, in Section 4.7, we present the computational study, and our conclusions in Section 4.8.

4.2 Definitions and notation

A MOILP with n variables, p objectives, and m constraints is written as follows:

$$P: \quad \min\{z(x) : x \in \mathcal{X}\}$$

where the p objective functions $z(x) = Cx$ are defined by a $p \times n$ matrix of objective coefficients C. The *feasible set* $\mathcal{X} = \{x \in \{0,1\} : Ax \geqq b\}$ is given by a $m \times n$ matrix of constraint coefficients A, and a right-hand-side vector b of size m. The image of the feasible set in the objective space is $\mathcal{Y} := C\mathcal{X} = \{Cx : x \in \mathcal{X}\}$.

Since the objective function is vector-valuated, new operators need to be introduced to compare solutions. Let $y^1, y^2 \in \mathcal{Y}$, y^1 *weakly dominates* y^2 ($y^1 \leqq y^2$) if $y^1_k \leq y^2_k$ for all $k \in \{1, ..., p\}$. Besides, y^1 *dominates* y^2 ($y^1 \leqslant y^2$) if $y^1 \leqq y^2$ and $y^1 \neq y^2$. These relations also extend to sets of points. Let $\mathcal{S}^1, \mathcal{S}^2 \subset \mathbb{R}^p$, we say that $\mathcal{S}^1$ *dominates* $\mathcal{S}^2$ if for all $s^2 \in \mathcal{S}^2$, there exists $s^1 \in \mathcal{S}^1$ such that $s^1 \leqslant s^2$. The set $\mathcal{S}^1$ *partially dominates* $\mathcal{S}^2$ if $\mathcal{S}^1$ does not dominates $\mathcal{S}^2$, but there is at least one $s^2 \in \mathcal{S}^2$ such that there exists $s^1 \in \mathcal{S}^1$ such that $s^1 \leqslant s^2$.

Derived from the dominance relations, the *set of non-dominated points* is defined as $\mathcal{Y}_N = \{y \in \mathcal{Y} : \nexists y' \in \mathcal{Y}, y' \leqslant y\}$. This notation can be extended to any set $\mathcal{S} \subset \mathbb{R}^p$, i.e. $\mathcal{S}_N = \{y \in \mathcal{S} : \nexists y' \in \mathcal{S}, y' \leqslant y\}$. Moreover, we define the set $\mathbb{R}^p_{\geqq} = \{y \in \mathbb{R}^p : y \geqq 0\}$.

Given a set $\mathcal{S} \subset \mathbb{R}^p$, Ehrgott and Gandibleux (2007) introduced the notions of lower and upper bound sets for $\mathcal{S}_N$, which extend the concept of lower and upper bound to the multi-objective case. Let $\mathcal{S}^1, \mathcal{S}^2 \subset \mathbb{R}^p$, we define the operation $\mathcal{S}^1 + \mathcal{S}^2$ as the Minkowski sum, i.e., $\mathcal{S}^1 + \mathcal{S}^2 = \{s^1 + s^2 : s^1 \in \mathcal{S}^1, s^2 \in \mathcal{S}^2\}$. Moreover, $\mathcal{S}^1$ is $\mathbb{R}^p_{\geqq}$-closed if $\mathcal{S}^1 + \mathbb{R}^p_{\geqq}$ is closed, and $\mathbb{R}^p_{\geqq}$-bounded if there exists $s \in \mathbb{R}^p_{\geqq}$ such that $\mathcal{S}^1 \subset \{s\} + \mathbb{R}^p_{\geqq}$. The definition of Ehrgott and Gandibleux (2007) is recalled in Definition 4.1.

Definition 4.1. (Ehrgott and Gandibleux, 2007) Let $\mathcal{S} \subset \mathbb{R}^p$.

- A lower bound set $\mathcal{L}$ for $\mathcal{S}_N$ is an $\mathbb{R}^p_{\geqq}$-closed and $\mathbb{R}^p_{\geqq}$-bounded set such that $\mathcal{S}_N \subset \mathcal{L} + \mathbb{R}^p_{\geqq}$ and $\mathcal{L} = \mathcal{L}_N$.
- An upper bound set $\mathcal{U}$ for $\mathcal{S}_N$ is an $\mathbb{R}^p_{\geqq}$-closed and $\mathbb{R}^p_{\geqq}$-bounded set such that $\mathcal{S}_N \subset \text{cl}[\mathbb{R}^p \backslash (\mathcal{U} + \mathbb{R}^p_{\geqq})]$ and $\mathcal{U} = \mathcal{U}_N$, where cl(.) denote the closure operator.

A particular lower bound set is the singleton $\{y^I\}$ where y^I, called the *ideal point*, is defined by $y^I_k = \min_{y \in \mathcal{Y}_N}\{y_k\}$. Similarly, one can define the upper bound set $\{y^N\}$ where y^N, the *nadir point*, is such that $y^N_k = \max_{y \in \mathcal{Y}_N}\{y_k\}$.

The linear relaxation of a MOILP P is the problem P^{LP} : $\min\{z(x) : x \in \mathcal{X}^{LP}\}$, where $\mathcal{X}^{LP} = \{x \in [0,1] : Ax \geqq b\}$. The problem P^{LP} belongs to the class of *Multi-Objective Continuous Linear Problem* (MOCLP). Ehrgott and Gandibleux (2007) showed that solving the linear relaxation yields a valid lower bound set.

Given an upper bound set $\mathcal{U}$, Klamroth et al. (2015) proposed an alternative description of the region $\text{cl}[\mathbb{R}^p \backslash (\mathcal{U} + \mathbb{R}^p_{\geqq})]$ using the *set of local upper bounds* $\mathcal{N}(\mathcal{U})$. Let $u \in \mathbb{R}^p$, we define $\mathcal{C}(u) = \{z \in \mathbb{R}^p : z \leqq u\}$. Using the definition of Klamroth et al. (2015), the set of local upper bounds $\mathcal{N}(\mathcal{U})$ is the set such that $\bigcup_{u \in \mathcal{N}(\mathcal{U})} \mathcal{C}(u) = \text{cl}[\mathbb{R}^p \backslash (\mathcal{U} + \mathbb{R}^p_{\geqq})]$, and for all $u^1, u^2 \in \mathcal{N}(\mathcal{U})$, $u^1 \neq u^2$, $\mathcal{C}(u^1) \nsubseteq \mathcal{C}(u^2)$. The first condition makes sure that $\mathcal{N}(\mathcal{U})$ describes properly $\text{cl}[\mathbb{R}^p \backslash (\mathcal{U} + \mathbb{R}^p_{\geqq})]$, whereas the second condition implies that there is no pair of local upper bounds such that one dominates the other, i.e., $\mathcal{N}(\mathcal{U})$ is of minimal size.

Given a lower bound set $\mathcal{L}$ and an upper bound set $\mathcal{U}$, we define the search region as the set $\mathcal{L} + \mathbb{R}^p_{\geqq} \cap \text{cl}[\mathbb{R}^p \backslash (\mathcal{U} + \mathbb{R}^p_{\geqq})]$. The search region can be interpreted as the region of the objective space where non-dominated points are possibly located.

A *weighted-sum scalarization* P_λ of a MOILP P is a single-objective optimization problem where the objective function is a weighted sum of the objective functions of P. Hence, given a weight vector $\lambda \in \mathbb{R}^p_{\geqslant}$, the problem P_λ is written as $P_\lambda : \min\{\lambda z(x) : x \in \mathcal{X}\}$.

Property 4.1. *(Ehrgott, 2005) Let P be a MOCLP. Its non-dominated set $\mathcal{Y}_N$ corresponds to the non-dominated part of a polyhedron, and for any weighted-sum scalarization P_λ with weight $\lambda \in \mathbb{R}_{\geqslant}$, there is an extreme point of $\mathcal{Y}_N$ that is optimal for P_λ.*

Given a MOILP P, since its linear relaxation P^{LP} is a MOCLP, Property 4.1 implies that the optimal solution of a weighted-sum scalarization of P^{LP} can be obtained by searching for the extreme point of the lower bound set that has the minimal weighted-sum of its objective values. This property will be exploited in Section 4.6.1.

4.3 Related work

To our knowledge, the first MOBB was proposed by Klein and Hannan (1982). In their algorithm, the authors use a single branching tree to solve a series of single-objective problems to generate all desired solutions. Later, Kiziltan and Yucaoğlu (1983) proposed a framework that uses the minimal completion, which consists of setting variables to 0 or 1 depending on its objective coefficients, to generate lower bounds. The resulting solution is integer but is not necessarily feasible for the initial problem.

In the following decade, a lot of attention was paid to DSS approaches tailored to specific problems. We refer the reader to Ramos et al. (1998) and Visée et al. (1998) for studies on the minimum Spanning Tree problem and the Knapsack problem, respectively. In the latter, the novelty lies in the fact that they use a branch-and-bound algorithm in the second phase of a two-phase method, a well-known OSS algorithm proposed by Ulungu and Teghem (1995). In other words, they embedded a DSS algorithm into an OSS algorithm, resulting in the first method that effectively mixes both approaches.

In multi-objective optimization, the ideal point provides a straightforward lower bound set. The first to introduce more complex lower bounds in a DSS algorithm are Sourd and Spanjaard (2008). In their paper, the authors use a surface as a lower bound set, namely the convex relaxation. Thereafter, the linear relaxation, weighted-sum scalarizations, and the linear relaxations of weighted sum scalarizations have been widely used in a similar way (see e.g., Vincent et al. (2013), Stidsen et al. (2014), Belotti et al. (2016), Stidsen and Andersen (2018), Parragh and Tricoire (2019), Gadegaard et al. (2019), Adelgren and Gupte (2022)). Although all these studies focus on the bi-objective case, the separating hypersurface principle from Sourd and Spanjaard (2008) is also applicable in higher dimensions. Recently, Santis et al. (2020) generated hyperplanes to obtain lower bound sets for multi-objective convex optimization problems (thus including MOILP) with three or more objective functions, whereas Forget et al. (2022) proposed to solve the linear relaxation for MOILP using more than two objectives. In the latter, the authors emphasized the difficulties raised by adding a third objective function. Indeed, if a simple dichotomic search is sufficient to calculate the linear relaxation with two objectives, things become more difficult when more dimensions are considered. In their paper, the authors suggest using Benson's outer approximation algorithm (Benson, 1998; Hamel et al., 2013; Löhne and Weißing, 2020) to compute the linear relaxation based lower bound set.

Multi-Objective Mixed-Integer Linear Problems (MOMILP), i.e., problems with both continuous and integer variables, have also received some attention in the MOBB literature. Mavrotas and Diakoulaki (1998) proposed a branch-and-bound framework that can handle MOMILP, as well as an improved version of their algorithm in Mavrotas and Diakoulaki (2005). Later, Vincent et al. (2013) proposed a refined version of their framework for the bi-objective 0-1 case. The use of MOBB to solve bi-objective MOMILP was further studied by Belotti et al. (2016) and Adelgren and Gupte (2022).

In their paper, Vincent et al. (2013) also conducted a study of different node selection rules. They tested depth-first and breadth-first strategies on randomly generated instances, and depth-first appeared to be the most efficient. Similarly, Parragh and Tricoire (2019) tested both approaches on a different set of instances, but breadth-first performed the best. This suggests that the performance of the two classical node selection rules used in the literature, namely depth-first and breadth-first, are, in fact, dependent on the problem class. A similar observation was made in the preliminary study of Forget et al. (2022), where both rules resulted in very different CPU times depending on the problem class of the instance solved. This issue is addressed in Sections 4.6 and 4.7.

In the past decade, a lot of attention has been paid to methods that hybridize DSS and OSS algorithms. For the bi-objective case, Stidsen et al. (2014) proposed partitioning the objective space into multiple slices, leading to stronger upper bound sets. This also opened the door to parallelization, which was exploited in (Stidsen and Andersen, 2018), and resulted in promising improvements in performance. The authors also developed the concept of *Pareto branching* (or *objective (space) branching*): when the upper bound set partially dominates the lower bound set, it is possible to create disjoint sub-problems in the objective space by adding upper bounds on the objective functions to discard dominated regions from the search. This principle was further explored and improved independently by Gadegaard et al. (2019) and Parragh and Tricoire (2019). In both papers, their experiments showed the great efficiency of this technique for the bi-objective case. Later, Adelgren and Gupte (2022) also showed promising results using objective branching in MOBB applied to bi-objective MOMILP.

Forget et al. (2022) extended objective branching to the multi-objective case. In their paper, the authors highlighted several challenges that arise when three or more objective functions are considered. In particular, they established that generating sub-problems without redundancies is a much more complex task compared to the bi-objective case, and they proposed a new method to overcome these difficulties. As a consequence, although still beneficial, objective branching did not appear to be as efficient as in the bi-objective case in their computational study. In this paper, we improve this result by showing that combining probing with objective branching results in a significant speed-up.

Recently, Adelgren and Gupte (2022) have proposed to use probing, a popular technique used in the single-objective case, to enhance their bi-objective branch-and-bound framework. The probing procedure of Adelgren and Gupte (2022) relies on solving the bi-objective linear relaxation based bound set, and showed promising results in their experiments. However, the impact of probing for problems with three or more objectives in unclear, as bound sets are more complex to compute. Furthermore, objective branching cannot be applied as often and easily as in the bi-objective case, which may also have an impact on the performance of probing.

4.4 Branch-and-bound framework

The branch-and-bound framework developed in this paper is based on the framework of Forget et al. (2022), and is presented in this section. Similarly to the single-objective case, the principle is to divide a problem that is too hard to be solved into easier sub-problems. Each sub-problem is stored in a node, and the nodes are grouped together into a tree

```
1: Create the root node $\eta^0$; set $\mathcal{T} \leftarrow \{\eta^0\}$ and $\mathcal{U} \leftarrow \emptyset$
2: while $\mathcal{T} \neq \emptyset$ do
3:     Select a node $\eta$ from $\mathcal{T}$; Set $\mathcal{T} \leftarrow \mathcal{T} \backslash \{\eta\}$
4:     Compute a local lower bound set for $P(\eta)$
5:     If possible, update the upper bound set $\mathcal{U}$
6:     if $\eta$ cannot be fathomed then
7:         Split $P(\eta)$ into disjoint subproblems $P(\eta^1), ..., P(\eta^h)$, and store each in a
           unique children node of $\eta$.
8:     end if
9: end while
10: return $\mathcal{U}$
```

Algorithm 4.1: Branch-and-bound algorithm for MOCOs

data structure. For each node η, the sub-problem contained in η is called $P(\eta)$, and each subproblem of $P(\eta)$ is stored in a child node of η. Instead of single numerical values, lower and upper bound sets are used to determine whether a given sub-problem potentially contains new non-dominated solutions, which are feasible for the initial problem. If not, the corresponding node is fathomed. Otherwise, it is divided into disjoint sub-problems. A general outline of our framework is presented in Algorithm 4.1.

The branch-and-bound is initialized with an empty upper bound set $\mathcal{U}$, and a list of non-explored nodes, denoted by $\mathcal{T}$, that contains the initial problem (at the root node of the tree). At each iteration, a node η is selected and removed from $\mathcal{T}$ (Line 3 of Algorithm 4.1). The classical tree exploration strategies from the literature are depth-first (last in first out) and breadth-first (first in first out), and are further discussed in Section 4.7. If there is no non-explored node remaining, the algorithm stops and $\mathcal{U} = \mathcal{Y}_N$.

When a node η is selected, a lower bound set $\mathcal{L}(\eta)$ for $P(\eta)$ is computed (Line 4 of Algorithm 4.1). In this framework, the linear relaxation $P^{LP}(\eta)$ is solved, and the result yields a valid lower bound set (Ehrgott and Gandibleux, 2007). A Benson-type algorithm is used for this purpose (see, e.g., Hamel et al. (2013)). As a result, a description of $\mathcal{L}(\eta)$ in terms of its extreme points is obtained, as well as a description of $\mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}$ in terms of its hyperplanes.

Once the lower bound set is obtained, if possible, new non-dominated points are harvested (Line 5 of Algorithm 4.1). Indeed, Benson's algorithm returns a pre-image for each extreme point of $\mathcal{L}(\eta)$. Hence, any extreme point l with an integer pre-image that is not dominated by any existing point in the upper bound set will be added to $\mathcal{U}$; and all points $y \in \mathcal{U}$ that are dominated by l ($l \geq y$) are removed from $\mathcal{U}$.

We distinguish three cases in which a node can be fathomed. (i) If $P^{LP}(\eta)$ is infeasible, no new non-dominated points are searched (i.e., Line 5 is skipped), and the node is *fathomed by infeasibility*. Indeed, similarly to the single-objective case, if $P^{LP}(\eta)$ is infeasible, $P(\eta)$ is also infeasible.

(ii) If $\mathcal{L}(\eta)$ is made of a unique extreme point l with an integer pre-image, all new points found in $P(\eta)$ will be dominated by the integer solution l and consequently, η is *fathomed by optimality*.

(iii) Finally, a third way to fathom a node exists: *fathoming by dominance*. This case happens when the lower bound set $\mathcal{L}(\eta)$ is dominated by the upper bound set $\mathcal{U}$. From the definitions, this situation is equivalent to saying that each feasible point of $P^{LP}(\eta)$, and thus of $P(\eta)$, is dominated by at least one already known integer point $u \in \mathcal{U}$. Consequently, no new non-dominated point can be found in $P(\eta)$. In practice, if there exists no local upper bound $u \in \mathcal{N}(\mathcal{U})$ such that $u \in \mathcal{L}(\eta) + \mathbb{R}^p_{\geqslant}$, then the node is fathomed by dominance. This dominance test was first introduced by Sourd and Spanjaard (2008), and used multiple times in the literature (see e.g., Gadegaard et al. (2019); Forget et al. (2022)).

If the node η cannot be fathomed, we resort to *branching*, and $P(\eta)$ is split into several sub-problems (Line 7 of Algorithm 4.1). To do so, objective branching is used first. This technique was initially introduced for the bi-objective case by Stidsen et al. (2014), further improved by Gadegaard et al. (2019) and Parragh and Tricoire (2019), and finally extended to the multi-objective case by Forget et al. (2022). It consists of creating disjoint sub-problems in the objective space when the lower bound set is partially dominated by the upper bound set, with the purpose of discarding regions that cannot contain any new non-dominated points. The subproblems are created by adding constraints in the form $z(x) \leqq s$, $s \in \mathbb{R}^p$, and the point s is called *super local upper bound*. Objective branching will be further elaborated upon in Section 4.5.

Once objective branching is applied, a set of subproblems $\eta^1, ..., \eta^\gamma$ is obtained. Note that only one subproblem is obtained ($\gamma = 1$) if it is not possible to create two or more disjoint subproblems in the objective space. Then, for each of the γ subproblems, *decision space branching* is performed. To do so, one variable x_i is chosen, and two subproblems with the constraints $x_i = 0$ or $x_i = 1$ are created. The variable x_i has to be a free variable, i.e., not fixed to a specific value in the current node by one of the previous branching decisions.

4.5 Enhanced objective branching

In this section, we further explore the concept of objective branching. Figure 4.1 depicts a situation where the lower bound set $\mathcal{L}(\eta)$ is partially dominated by the upper bound set $\mathcal{U}$, and the resulting search region is given by the hashed areas. Any part of the objective space that is not included in one of these areas cannot contain any feasible non-dominated

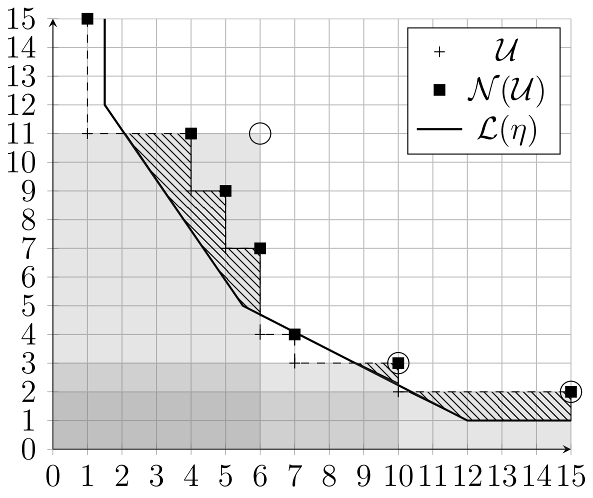


Figure 4.1: The lower bound set $\mathcal{L}(\eta)$, depicted by the solid line, is partially dominated by the upper bound set $\mathcal{U}$, represented by the crosses. In this situation, it is possible to split the problem into three disjoint sub-problems in the objective space. Each sub-problem is highlighted by the hatched areas, and its corresponding super local upper bound is depicted by its closest large circle.

point for $P(\eta)$. Objective branching consists of generating disjoint subproblems in a way such that as much of the region of the objective space dominated by $\mathcal{U}$ is discarded from all subproblems, without excluding any point of the search region from the sub-problems. In the example from Figure 4.1, this resulted in three subproblems, defined by three super local upper bounds depicted by the large circles.

In the bi-objective case, multiple ways to compute the subproblems exist. Stidsen et al. (2014) and Gadegaard et al. (2019) generated new subproblems when the algorithm detected that one or several points of the upper bound set partially dominate the lower bound set, whereas Parragh and Tricoire (2019) kept track of the various non-dominated segments of the lower bound set and generated a subproblem for each. The two approaches are equivalent in the sense that exactly the same subproblems are generated with both methods.

Recently, Forget et al. (2022) showed that the computation of the subproblems in the case where $p \geq 3$ was more complex but still possible. However, the increased complexity resulted in a less significant benefit of using objective branching when $p \geq 3$ compared to the case where $p = 2$. For some problem classes, it even resulted in worse computation times, which creates a great contrast with the bi-objective case, where using objective branching systematically led to lower CPU time. As a result, it appears that when $p \geq 3$, the use of objective branching is not always sufficient by itself, and in this paper, we aim to study whether objective branching constraints can be further exploited to help reducing the total CPU time of the branch-and-bound framework.

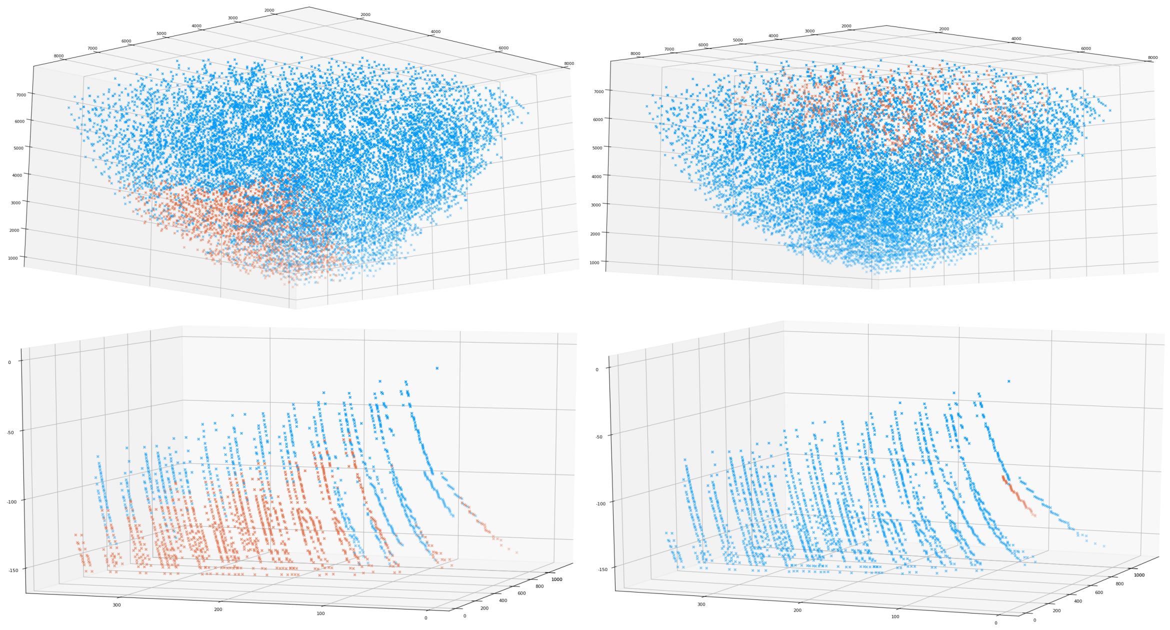


Figure 4.2: The first row depicts the set of non-dominated points for a tri-objective Uncapacitated Facility Location Problem with 4672 non-dominated points. The second row depicts the set of non-dominated points for a tri-objective Capacitated Facility Location Problem with 1912 non-dominated points. For each plot, a variable x_i was chosen. A blue point is a non-dominated point where $x_i = 0$ whereas an orange point is a non-dominated point for which $x_i = 1$.

4.5.1 Probing

In multi-objective optimization, an intuitive belief is that two points that are close to each other in the objective space are more likely to have similar pre-images than two points that are far away from each other. Figure 4.2 shows the set of non-dominated points of two instances of tri-objective MOCO instances (one row for each instance). A blue point corresponds to a non-dominated point where the chosen variable x_i takes value 0, whereas an orange point corresponds to a non-dominated point such that $x_i = 1$. From these four pictures, it is clear that some problem classes have variables that take a particular value in certain parts of the objective space. Moreover, when applying objective branching, the algorithm reduces the search to particular regions of the objective space. Hence, based on the previous observation, it is possible that some variables cannot take specific values anymore in certain subproblems. The process of identifying such values is commonly referred to as probing. In the following, we present our probing strategies.

A naive strategy

We explore at first a naive strategy. At node η, we define the set of variables fixed to 0 and 1 as $\mathcal{I}^0(\eta) = \{i \in \{1, ..., n\} : x_i = 0\}$ and $\mathcal{I}^1(\eta) = \{i \in \{1, ..., n\} : x_i = 1\}$ respectively. The set of free variables is $\mathcal{I}^f(\eta) = \{1, ..., n\} \backslash (\mathcal{I}^0(\eta) \cup \mathcal{I}^1(\eta))\}$. With a little abuse of notation, we will consider that writing $x_i \in \mathcal{I}^0(\eta)$ is equivalent to $i \in \mathcal{I}^0(\eta)$. We will consider analogous statements for $\mathcal{I}^1(\eta)$ and $\mathcal{I}^f(\eta)$ as well.

Let $x_i \in \mathcal{I}^f(\eta)$ be a free variable at node η. Since we consider problems with binary variables only, the possible values for x_i are 0 or 1. A first approach in order to check whether x_i can take value $t \in \{0, 1\}$ in $P(\eta)$ is to solve the linear program $F(i, t)$: $\min\{0 \mid x \in \mathcal{X}^{LP}(\eta),\ x_i = t\}$. If $F(i, t)$ is not feasible, then x_i cannot take value t. When both $F(i, 0)$ and $F(i, 1)$ are solved for x_i, there are four possible scenarios:

- Both $F(i, 0)$ and $F(i, 1)$ are feasible: nothing can be concluded about x_i, and thus, the variable remains a free variable;
- $F(i, 0)$ is feasible and $F(i, 1)$ is infeasible: x_i is fixed to 0;
- $F(i, 0)$ is infeasible and $F(i, 1)$ is feasible: x_i is fixed to 1;
- Both $F(i, 0)$ and $F(i, 1)$ are infeasible: there is no possible integer value for x_i. Thus, the node η is fathomed by infeasibility.

When x_i is fixed to 0, the set of free variables $\mathcal{I}^f(\eta)$ is updated to $\mathcal{I}^f(\eta) \backslash \{i\}$ since x_i is not free anymore, and $\mathcal{I}^0(\eta)$ becomes $\mathcal{I}^0(\eta) \cup \{i\}$. Similarly, if x_i is fixed to 1, then $\mathcal{I}^f(\eta)$ and $\mathcal{I}^1(\eta)$ become $\mathcal{I}^f(\eta) \backslash \{i\}$ and $\mathcal{I}^1(\eta) \cup \{i\}$ respectively. If all free variables are fixed to a particular value, the node is fathomed by optimality. Indeed, this situation implies that there is only one integer solution in $P(\eta)$, and no new non-dominated point can be reached in $P(\eta)$. The upper bound set is updated with the new point obtained by fixing all variables.

A first naive strategy is to solve $F(i, t)$ at each node η, for each free variable $x_i \in \mathcal{I}^f(\eta)$, and for each possible value $t \in \{0, 1\}$. The approach is similar to Adelgren and Gupte (2022) in the sense that they also perform probing at each node. In their paper, the authors suggest to perform probing both before the computation of the linear relaxation, and when creating sub-problems. In the latter case, they apply probing after selecting a free variable to branch on, and change the branching variable if they conclude that the decision led to an infeasible problem. In this paper, we adopted a slightly different approach: we perform probing only after objective branching and before variable branching. In this way, we aim to reduce the set of branching candidates at each node, while still benefiting from the objective branching constraints. Indeed, we expect these to be the most constraining to the problem, since they restrict the search to a particular region of the objective space and thus, hopefully, reduce the possible values taken by the variables and help the algorithm to make an appropriate branching decision.

Another difference with Adelgren and Gupte (2022) is that when performing probing, they solve the bi-objective linear relaxation of the corresponding problem instead of solving a simple feasibility problem as we do. However, as we consider more objective functions, the linear relaxation becomes significantly more expensive to compute. Consequently, considering that we only check each variable once, our approach requires at most one single-objective linear program to be solved for each variable and value, which, in the binary case, limits the maximum number of linear programs to be solved to $2\mathcal{I}^f(\eta)$ at each node.

An advanced strategy

The naive strategy can be improved. Indeed, it is possible in some cases to detect if $F(i,t)$ is feasible without actually solving the linear program. For instance, if a solution that is feasible for $F(i,t)$ is already known, there is no need to solve $F(i,t)$. Such solutions can be collected, for example, from the extreme points of the lower bound set, or by keeping track of the solutions obtained from previously solved linear programs ($F(j,t')$, $j \neq i$, $t' \in \{0,1\}$) in the current node.

Moreover, variables can be fixed by inspection. Let $x_j \in \mathcal{I}^f(\eta)$ be a free variable, and $\sum_{l=1}^{n} a_{il}x_l \leq b_i$ a constraint of the problem. By comparing a_{ij} to the maximal possible value of the right-hand side of the constraint, one may be able to conclude that x_i cannot take value 1. First, all variables fixed to 1 are considered as constants and will be used to adjust the right-hand side. Then, all free variables $x_l \in \mathcal{I}^f(\eta)$ such that $a_{il} \leq 0$ will be temporarily fixed to 0 meanwhile, those where $a_{il} < 0$ will be temporarily fixed to 1. The constraint then becomes $a_{ij}x_j \leq b - \sum_{l\in\mathcal{I}^1(\eta)} a_{il} + \sum_{l\in\mathcal{I}^f(\eta),a_{il}<0,l\neq j} a_{il}$. Then, if $a_{ij} > b - \sum_{l\in\mathcal{I}^1(\eta)} a_{il} - \sum_{l\in\mathcal{I}^f(\eta),a_{il}<0,l\neq j} a_{il}$, the variable x_j can be fixed to 0. Similar rules can be used for constraints in the form $\sum_{l=1}^{n} a_{il}x_l \geq b_i$ and $\sum_{l=1}^{n} -a_{il}x_l \leq -b_i$. More complex preprocessing rules could be used as well, but this is out-of-scope of this paper and thus, we will stick to these simple rules here.

We note that probing may be applied after variable branching as well as after objective branching. However, our experiments showed that it is most effective when used in conjunction with objective branching. A possible explanation relates to the example given in Figure 4.2: some variables may only take certain values in certain regions of the objective space and objective branching induces such regions. Hence, at node η, we propose to apply probing only if an improvement in the objective branching constraints is observed compared to its parent node $\hat{\eta}$. In other words, we perform probing only if $s \leqslant \hat{s}$, where s and $\hat{s}$ are the super local upper bounds defining the objective branching constraints in nodes η and $\hat{\eta}$, respectively.

This advanced strategy is new compared to Adelgren and Gupte (2022), as we first aim at achieving the same results by solving fewerlinear programs, and then suggest using probing only when it is expected to be the most relevant.

Combining probing and bounding

Finally, the linear program $F(i,t)$ solved when applying the naive strategy does not have any objective function. However, using an objective function may provide us with additional information. For this purpose, we rely on adding a weighted sum objective function. Given a weight vector $\lambda \in \mathbb{R}^p$, the linear program $F(i,t,\lambda) : \min\{\lambda z(x) \mid x \in \mathcal{X}^{LP}(\eta),\ x_i = v\}$ is solved, and the optimal value z^{*t} is obtained. By definition, when x_i is fixed to t in this sub-problem, all feasible solutions $x \in \mathcal{X}(\eta) \cap \{x_i = t\}$ are such that $\lambda z(x) \geq z^{*t}$. Moreover, because of the objective branching constraints, all feasible solutions $x \in \mathcal{X}(\eta) \cap \{x_i = t\}$ are such that $z(x) \leqq s$, $s \in \mathbb{R}^p$. Hence, we can conclude that if there is no local upper bound $u \in \mathcal{N}(\mathcal{U})$ such that $\lambda u \geq z^{*t}$ and $u \leqq s$, then there is no feasible solution $x \in \mathcal{X}(\eta) \cap \{x_i = t\}$ that can generate a new non-dominated point. In this case, x_i is fixed to 1 if $t = 0$, or to 0 if $t = 1$. Note that the dominance test we employ here has, e.g., also been used by Stidsen et al. (2014); Stidsen and Andersen (2018) in a bi-objective context.

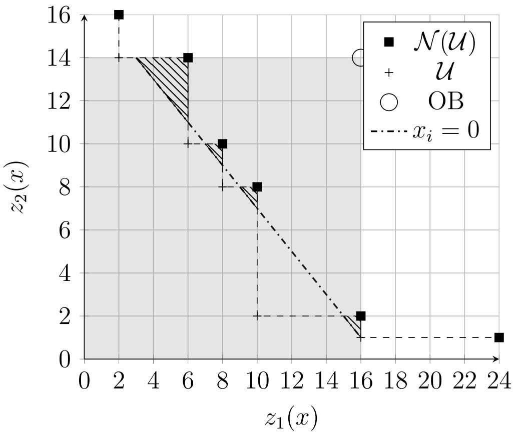
An example is given in Figure 4.3, where $\lambda = (1,1)$ is used, and the programs $F(i,0,\lambda)$ and $F(i,1,\lambda)$ are solved. In the rightmost figure, $F(i,0,\lambda)$ is feasible, and the weighted-sum resulted in a non-empty search region. On the contrary, in the leftmost figure, $F(i,1,\lambda)$ is feasible but the weighted-sum is dominated by the upper bound set in the region considered. This implies that in this sub-problem, all integer solutions in which x_i take value 1 are dominated by at least one existing integer solution. Hence, there is no need to branch on x_i, and the variable can be fixed to 0.

This strategy is closer to the one proposed by Adelgren and Gupte (2022) for the bi-objective case in the sense that we also compute a lower bound set, namely the linear relaxation of a weighted-sum scalarization. However, our lower bound set is weaker than theirs (linear relaxation), but requires only one linear program to be solved.

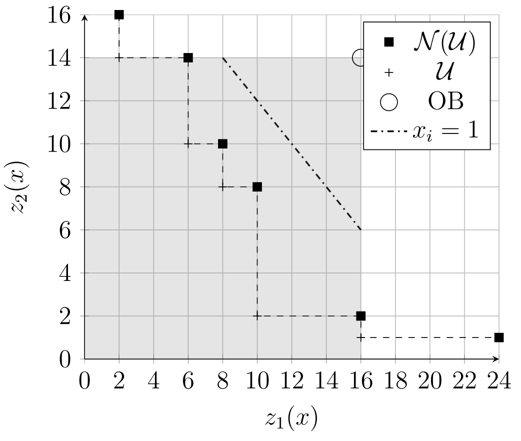
4.5.2 Objective branching based cover inequalities

For an objective k that is minimized, an objective branching constraint has the form $w^T x \leq b$, where w is given by the objective coefficients of $z_k(x)$, and b is derived from the bound on objective k in the sub-problem at hand. One can observe that this constraint is in fact a Knapsack constraint, from which cover inequalities can be derived (see, e.g., Gu, Nemhauser, and Savelsbergh (1998)).

By nature, objective branching constraints are often expected to be binding constraints, as they are included to create disjoint sub-problems. Hence, for an objective k such that the objective branching constraint $z^k \leq s^k$ is binding, there will be extreme points in the lower bound set whose k^{th} component will be equal to s^k. Thus, in case such an extreme point is fractional, cover cuts can be generated to cut it from the lower bound set, with the aim to



(a) The linear program $F(i, 0, \lambda)$ was solved, and the optimal value z^{*0} was obtained. All points of the objective space satisfying the objective branching constraints and $\lambda z(x) = z^{*0}$ are depicted by the dash-dotted line (weighted sum value). All feasible points are located both in the gray area (objective branching constraints) and above the dashdotted line. Since some local upper bounds are located in this region, $x_i = 0$ is a possible candidate for new non-dominated points in this region of the objective space.



(b) The linear program $F(i, 1, \lambda)$ was solved, and the optimal value z^{*1} was obtained. All points of the objective space satisfying the objective branching constraints and $\lambda z(x) = z^{*1}$ are depicted by the dash-dotted line. All feasible points are located both in the gray area (objective branching constraints) and above the dashdotted line(weighted sum value). Since no local upper bounds are located in this region, $x_i = 1$ is not a possible value for new non-dominated points in this region of the objective space.

Figure 4.3: Both $F(i, 0, \lambda)$ and $F(i, 1, \lambda)$ are solved with $\lambda = (1, 1)$, resulting in the situations depicted in the left and right figures respectively. Given the objective branching constraints, it is concluded that x_i cannot take value 1 and thus, x_i is fixed to 0.

move it closer to an integer point.

Let $l \in \mathcal{L}(\eta)$ be an extreme point such that $l_k = s_k$, and $x^l \in \mathcal{X}^{LP}(\eta)$ its pre-image. We define $\hat{\mathcal{J}}(l) = \{j \in \{0, ..., n\} : x^l_j = 1\}$ as the set of indices of the variables that take value 1 in x^l, and $\bar{\mathcal{J}}(l) = \{j \in \{0, ..., n\} : 0 < x^l_j < 1\}$ as the set of indices of the variables that take a fractional value in x^l. By definition, $\sum_{j \in \hat{\mathcal{J}}} c^k_j x_j + \sum_{j \in \bar{\mathcal{J}}} c^k_j x_j = s^k$ holds true. If x^l is fractional, i.e., $\bar{\mathcal{J}} \neq \emptyset$, then $\sum_{j \in \hat{\mathcal{J}}} c^k_j + \sum_{j \in \bar{\mathcal{J}}} c^k_j < s^k$ also holds true because for all $j \in \bar{\mathcal{J}}$, we have $x_j < 1$. Hence, all variables x_j such that $j \in \hat{\mathcal{J}} \cup \bar{\mathcal{J}}$ cannot simultaneously take value 1. Thus, $\sum_{j \in \hat{\mathcal{J}} \cup \bar{\mathcal{J}}} x_j \leq |\hat{\mathcal{J}} \cup \bar{\mathcal{J}}| - 1$ is an example of a cover inequality that can be generated from x^l.

Of course, in many cases, different cover inequalities can be generated. These cuts can also be strengthened by using any of the available lifting procedures from the literature.

4.6 Node selection rules

In the MOBB literature, breadth-first and depth-first are the commonly employed node selection rules. Indeed, the fact that these rules are independent from the nature of the problem being solved constitutes a good reason to use such rules when expanding branch-and-bound methods to the multi-objective case, as they do not require additional work to make the algorithm perform correctly. However, previous studies have shown that depth-first is significantly better for some problem classes, whereas breadth-first is better for others (see e.g., Vincent et al. (2013); Parragh and Tricoire (2019); Forget et al. (2022)). This inconsistency is problematic when building a generic solver as we do here, since it could easily lead to very poor performances in some cases. Therefore, we aim to explore alternative node selection rules in the hope of finding rules that are more robust across problem classes, and that perform better than the classical depth and breadth-first strategies.

In the single objective literature, the so-called best-bound strategy (and variations thereof) has shown to be of value (Linderoth and Savelsbergh, 1999). Its basic principle consists in exploring first the node that has the lowest lower bound value, as it constitutes the most promising area of the decision space. Unfortunately, in the multi-objective case, it is often not a trivial task to determine which node has the best bound, since one may have a lower bound set that is better than the others in a particular region of the objective space, but worse in other regions.

In the remainder of this Section, we propose two rules based on the best-bound principle. In Section 4.6.1, we present a rule that searches for the best bound in a specific part of the objective space by using weighted-sum values. In Section 4.6.2, we define a rule that is based on gap measures between lower and upper bound sets.

4.6.1 Weighted-sum rule

A straightforward way to mimic the best-bound approach in a MOBB is to consider a weighted-sum scalarization, and to use the value of its linear relaxation as a measure of the quality of the lower bound set in each node. Let η be a node of the tree, let λ be the weight vector used for the scalarization $P_\lambda(\lambda)$, and z^* the optimal value of its linear relaxation $P_\lambda^{LP}(\lambda)$. The score $s(\eta)$ of the node η is then given by $s(\eta) = z^*$, and the node with the lowest score is selected.

Note that this rule is equivalent to a best-bound strategy using a branch-and-bound to solve the problem P_λ. Hence, translated in the context of MOBB, one can say that this strategy develops the node that is the most promising in direction λ first.

From a computational point of view, the score of a new node η has to be calculated at its creation, which, in the present framework, requires solving a single-objective linear

```
1: Create the root node $\eta^0$; set $\mathcal{T} \leftarrow \{\eta^0\}$ and $\mathcal{U} \leftarrow \emptyset$
2: while $\mathcal{T} \neq \emptyset$ do
3:    Select the node $\eta$ with the best score from $\mathcal{T}$; Set $\mathcal{T} \leftarrow \mathcal{T}\backslash\{\eta\}$
4:    if $\eta$ cannot be fathomed then
5:       Split $P(\eta)$ into disjoint subproblems $P(\eta^1), ..., P(\eta^h)$, and store each in a
         unique children node of $\eta$.
6:       for $\hat{\eta} \in \{\eta^1, ..., \eta^h\}$ do
7:          Compute a local lower bound set for $P(\hat{\eta})$
8:          Update the upper bound set $\mathcal{U}$
9:          Compute the score $s(\hat{\eta})$ for $\hat{\eta}$
10:      end for
11:   end if
12: end while
13: return $\mathcal{U}$
```

Algorithm 4.2: An alternative branch-and-bound algorithm for MOILPs using a best-bound strategy

program, namely $P^{LP}_{\lambda}(\eta)$. However, using a simple re-ordering of the steps of Algorithm 4.1, it is possible to obtain the score of η without solving a linear program. Indeed, it is well known that all points of the non-dominated set of a multi-objective continuous linear problem correspond to an optimal solution of a weighted-sum scalarization Ehrgott (2005). In our context, this implies that at node η, the score of η can be obtained by searching for the point $l^* \in \mathcal{L}(\eta)$ such that there is no other l for which $\lambda l < \lambda l^*$. In other words, we search for the point of the lower bound set with the minimum weighted-sum value given the weight vector λ. Fortunately, this point is given by an extreme point of $\mathcal{L}(\eta)$ (see Property 4.1), and only extreme points have to be checked. Hence, by computing the lower bound set at the creation of the node instead of when the node is selected, the score can be obtained at a very low cost. Note that whether the lower bound set is computed at the creation or at the selection of the node does not make a difference, as $P^{LP}(\eta)$ does not change. Furthermore, this result also holds for the update of the upper bound set, that only depends on the solutions found in the lower bound set. However, this is not true for fathoming, and in particular, fathoming by dominance. Indeed, new feasible points may be found between the creation and the selection of a node, which may allow the node to be fathomed by dominance. Hence, Lines 7 and 8 are moved to after the creation of the node, and the computation of the score is performed as well, which results in the new framework given by Algorithm 4.2.

To conclude, this rule is very cheap and easy to compute. However, the drawback is that it is very representative in one direction only and neglects other regions of the objective space. For example, by using the weight vector $\lambda = (1, ..., 1)$, the rule is likely to find good solutions that are well balanced across all objectives more rapidly than good solutions that are very good in one of the objectives but bad for other objectives. However the actual impact on the performance is unclear, and is further studied in Section 4.7.

4.6.2 Gap measure rule

Another way to adapt the idea of best-bound strategies to the multi-objective case is to compute a measure of the gap between the upper and lower bound set in each node. In this case, the node with the largest gap is explored first, as it describes either a promising area, or a region where very few feasible points have been discovered, and possibly many more remain to be discovered.

An intuitive way to compute the gap in a given node is to compute the hypervolume of the search area. However, it is well known that it is a costly and difficult operation, particularly when three or more dimensions are considered. Hence, alternative measures are necessary. When Ehrgott and Gandibleux (2007) introduced the concept of lower and upper bound sets, they also proposed a number of measures to compare the quality of lower and upper bound set. One of these measures is similar to the Hausdorff distance, and consists in computing the minimal distance between the two points from each set respectively that are the furthest away. Recently, this measure has been used by Adelgren and Gupte (2022) to compute gaps between bound sets in the context of bi-objective mixed-integer branch-and-bounds.

In our context, at node η, the Hausdorff distance between the upper bound set and the lower bound set is given by $\max_{u \in \mathcal{N}(\mathcal{U})} \min_{l \in \mathcal{L}(\eta)} d(u, l)$, where $d(u, l)$ is the distance between u and l. From an implementation point of view, only local upper bounds that are located above the lower bound set are considered, as they are the only ones that define the search region in η. If there is none, we consider that the node has a gap of 0. This approach is, in fact, analogous to the single objective case: as long as the lower and upper bound sets have not met entirely, the gap is strictly positive, and the node cannot be fathomed by dominance.

When multiple objectives are considered, a difficulty arises due to the fact that when a new feasible point u is added to the upper bound set $\mathcal{U}$, the gap in some nodes may change. In particular, a node η^1 with a smaller gap than that of η^2 may end up with a gap larger than that of η^2 after the update of $\mathcal{U}$. This implies that in order to identify the node with the best score, the gap has to be recomputed in all nodes whenever $\mathcal{U}$ changes. Unfortunately, this may be computationally expensive, as it is not rare that many nodes are open at a given iteration of the MOBB.

To reduce the computational burden, we rely on two simple properties. First, when a new point is added to the upper bound set, the gap in all nodes can only stay the same or decrease. The reason is that the lower bound sets remain unchanged and the upper bound set improves when a new feasible point is found. This implies that the search region shrinks:

```
1: found ← FALSE
2: while !found do
3:     Select the node $\eta^1$ with the largest gap from $\mathcal{T}$; Set $\mathcal{T} \leftarrow \mathcal{T}\backslash\{\eta\}$
4:     Compute $g^{new}(\eta^1)$
5:     Select the node $\eta^2$ with the largest gap from $\mathcal{T}$
6:     if $g^{new}(\eta^1) \geq g^{old}(\eta^2)$ then
7:         found ← TRUE
8:     else
9:         $g^{old}(\eta^1) \leftarrow g^{new}(\eta^1)$
10:        $\mathcal{T} \leftarrow \mathcal{T} \cup \{\eta^1\}$
11:    end if
12: end while
13: return $\eta^1$
```

Algorithm 4.3: Selection of the node with the largest gap

the upper bound set moves closer to the lower bound sets. Second, we are only interested in the node with the largest gap, as it corresponds to the next node being explored. Let $\eta^1 \in \mathcal{T}$ be the node with the largest gap, and $\eta^2 \in \mathcal{T}$ be the node with the second largest gap. Let $g^{old}(\eta)$ be the gap of a node η before re-computation, and $g^{new}(\eta)$ be its gap after re-computation. By construction, we know that $g^{old}(\eta^2) \geq g^{new}(\eta^2)$. Furthermore, for all $\eta \in \mathcal{T}\backslash\{\eta^1, \eta^2\}$, we have $g^{old}(\eta^2) \geq g^{old}(\eta) \geq g^{new}(\eta)$. Hence, if $g^{new}(\eta^1) \geq g^{old}(\eta^2)$, only by re-computing the gap in η^1, we know that η^1 is the node with the largest gap after update of the upper bound set. If the condition is not satisfied, η^1 is put back in $\mathcal{T}$ and the process is repeated with η^2, the new potential node with the largest gap. The selection procedure is given in Algorithm 4.3.

4.7 Experiments

All algorithms are implemented in C++17, relying on Cplex 20.1 for solving single-objective linear programs, using a single thread. The experiments are carried out on Linux 10.3, on a Quad-core X5570 Xeon CPUs @2.93GHz processor and with 48GB of RAM memory. A time limit of one hour is set when running the algorithms.

Our computational study aims at answering the following research questions: (i) How does probing perform? In particular, how does it perform in combination with objective branching, and why? (ii) What is the impact of using an objective function in the linear programs used for performing probing? (iii) What is the impact of deriving cover cuts from the objective branching constraints on the performance of the algorithm? (iv) Can node selection rules based on the best-bound idea outperform the classical depth and breadth-first strategies often used in the literature? (v) By fixing variables, the set of potential candidates for branching is reduced. What is the impact of the chosen variable selection rule? (vi) Computing lower bound sets in the multi-objective case is expensive. Does resorting to pure enumeration at certain nodes in the tree improve the performance of the proposed MOBB? (vii) How does the proposed branch-and-bound framework performs in comparison to state-of-the-art objective space search algorithms?

We test our algorithms on the following three different types of problems and benchmark instances:

- Capacitated Facility Location Problems (CFLP). The instances are extracted from An, Parragh, Sinnl, and Tricoire (2022). Instances with 3 objectives 65, 230, and 495 variables are considered.
- Knapsack Problems (KP). The instances from Kirlik (2014) are used. Instances with 3 objectives and 40, 50, 60, 70, 80 variables are solved, as well as instances with 4 objectives and 20, 30, 40 variables.
- Uncapacitated Facility Location Problems (UFLP). The instances are extracted from Forget et al. (2022). For 3 objectives, instances with 56, 72, 90, 110 variables are used. For 4 objectives, instances with 42 and 56 variables are solved.

For each problem class, number of objectives, and number of variables, 10 instances are solved, leading to a total of 170 instances tested.

Various configurations are tested for the branch-and-bound algorithm in order to address the different research questions raised (objective branching, variable fixing, generation of cover cuts inequalities, node selection). The configurations tested will be clearly stated together with the results of each experiment. Unless specified otherwise, the framework uses the following parameters and heuristics:

- **Lower bound sets:** the linear relaxation is used as lower bound set. Its computation is warm-started by using the algorithm from Forget et al. (2022);
- **Local upper bounds:** the set of local upper bounds $\mathcal{N}(\mathcal{U})$ is updated whenever a new point is added to the upper bound set $\mathcal{U}$ by using the algorithm from Klamroth et al. (2015). Furthermore, all objective functions are expected to have integer coefficients, and all variables are binary. This implies that the non-dominated points can take integer values only. Consequently, when performing the dominance test and computing sub-problems through objective branching, each component of the local upper bound is shifted by -1.
- **Variable selection rule:** At the creation of sub-problems in the decision space, a free variable is chosen for branching. First, this variable is chosen independently in each

sub-problem obtained from objective branching. Let $s \in \mathbb{R}^p$ be the super local upper bound defining the sub-problem in which a free variable has to be chosen. The variable that is the most often fractional among the extreme points l of the lower bound set $\mathcal{L}(\eta)$ that satisfies $l \leqq s$ is selected. In case of ties, the one whose average value is closest to 0.5 is selected and in case of equal average values, the one with the smallest index is chosen.

Furthermore, unless specified otherwise, cover cut generation is disabled. Then, a number of different configurations are tested. The three main components evaluated in this study are the following:

- **Objective branching:** three options are considered: no objective branching (`NOB`); cone bounding (`CB`); and full objective branching (`FOB`), as presented in Section 4.4. Cone bounding is an alternative to objective branching proposed by Forget et al. (2020a). The idea is to derive upper bounds on the objective functions from the partial dominance of the lower bound set, but without splitting the objective space into sub-problems, i.e., only decision space branching is performed.
- **Probing/variable fixing:** three possibilities are considered: no variable fixing (`NVF`); variable fixing using the advanced strategy as presented in Section 4.5.1 (`VF`); and variable fixing using a weighted-sum objective function to allow for variable fixing by dominance (`VFD`) as explained in Section 4.5.1.
- **Node selection rule:** four configurations are considered: best of depth-first and breadth-first (`DB`); best-bound based on weighted sums (`BBWS`); a normalized version (`BBWSN`); and best-bound based on the gap measure presented in Section 4.6.2 (`BBGAP`). In the case of (`DB`), the best strategy per problem class is used. CFLP and UFLP use breadth-first, whereas KP uses depth-first. `BBWS` and `BBWSN` use $\lambda = (1, ..., 1)$ (see Section 4.6.1). Normalization may be important for problems for which the coefficients of the different objective functions take values in very different ranges, such as the CFLP.

In the remainder of this section, each sub-section is designed to address one of the research questions raised earlier.

4.7.1 Probing and objective branching

In a first step, we investigate the effect of combining probing with objective branching. We fix the node selection rule to configuration `DB`. First, six configurations are tested: the three objective branching strategies (`NOB`, `CB`, and `FOB`), in combination with two variable fixing configurations (`NVF` and `VF`). For `FOB` and `CB`, probing is performed only when an improvement

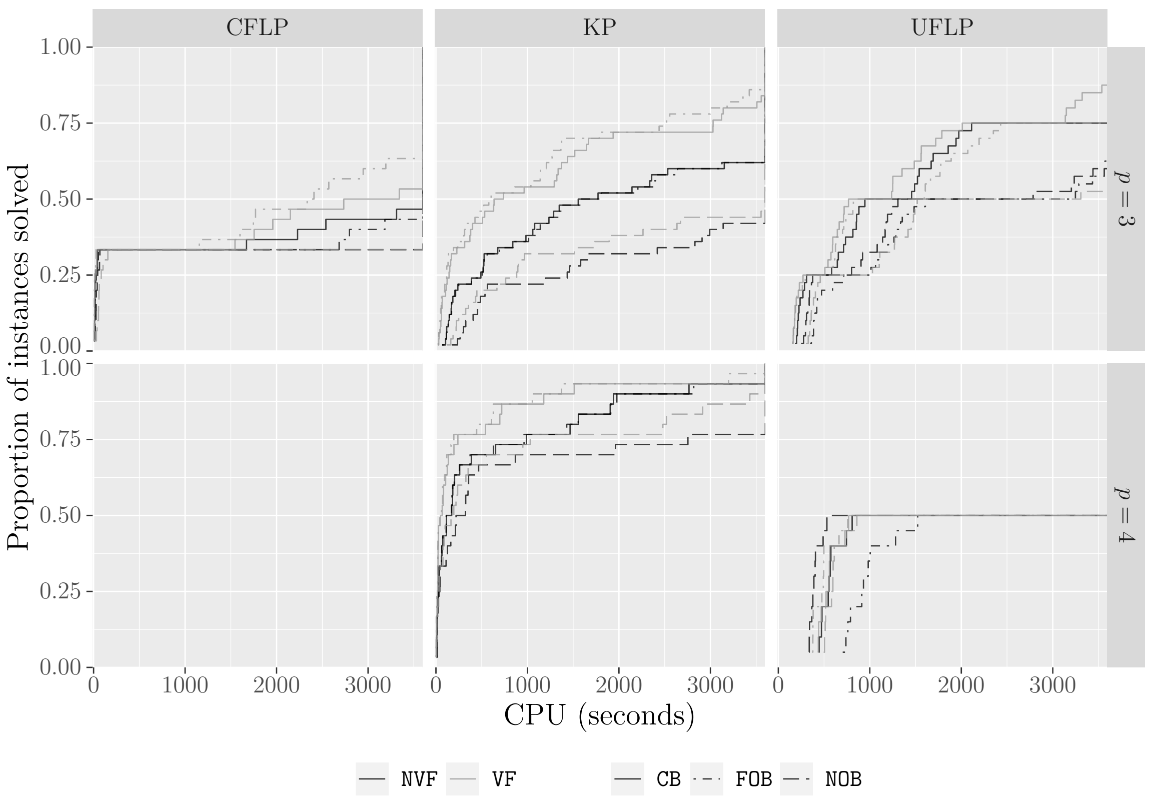


Figure 4.4: Performance curves of the different objective branching and probing configurations. The x-axis represents the CPU time expressed in second, and the y-axis corresponds to the proportion of instances solved. Each curve represents the proportion of instances solved through time for a given configuration. The line type indicates the objective branching configuration used, and the line color to the variable fixing configuration.

is observed in the objective branching constraints, as suggested in Section 4.5.1. However, probing is performed at every node for `NOB`.

In Figure 4.4, the performance profiles of the different configurations are given. The x-axis represents the CPU time expressed in seconds, whereas the y-axis corresponds to the proportion of instances solved. From this figure, it is clear that for both CFLP and KP, the winning configuration is `FOB-VF`. For UFLP and $p = 3$, however, although `FOB-VF` is the best configuration for smaller instances, `CB-VF` becomes the winning configuration for larger instances. For UFLP and $p = 4$, `NOB-NVF` is slightly faster than the other configurations.

It is interesting to note that enabling probing resulted in the greatest speed-ups for full objective branching (`FOB`). This is particularly striking for UFLP, where `FOB-NVF` is among the worst configurations, but `FOB-VF` is competitive with the best configurations. For CFLP, it is clear that `FOB` benefits more from probing than `CB` or `NOB`, since `CB-NVF` is faster than `FOB-NVF`, but `FOB-VF` is faster than `CB-VF`. This suggests that the branch-and-bound

			CB		FOB		NOB	
p	pb	n	NVF	VF	NVF	VF	NVF	VF
3	CFLP	65	7906.6 (0)	2382.6 (0)	11786.2 (0)	1389.4 (0)	15090.4 (0)	15072.4 (0)
		230	198405.7 (6)	65681.8 (4)	405597.4 (7)	49365.9 (1)	78613.1 (10)	32593.7 (10)
		495	6380.6 (10)	4357.9 (10)	33545.9 (10)	4497.9 (10)	4245.2 (10)	2934.4 (10)
	KP	40	104268.4 (0)	15918.4 (0)	136980.2 (0)	13023.6 (0)	137544.2 (0)	71502.4 (0)
		50	283955 (0)	34498.6 (0)	385177 (0)	27261.8 (0)	385625.2 (0)	174757.2 (0)
		60	804121.5 (1)	121213.2 (0)	1128878.7 (2)	95023.4 (0)	623624.4 (9)	428664.6 (7)
		70	1310720.2 (7)	289369.3 (3)	1839920.2 (7)	226918 (2)	617265.9 (10)	456791.8 (10)
		80	1368612.3 (10)	377936.3 (5)	1812818.1 (10)	303356 (5)	499527.9 (10)	367138.9 (10)
	UFLP	56	49212.2 (0)	20885.2 (0)	216010.6 (0)	14512.6 (0)	63239 (0)	62944.2 (0)
		72	111566.6 (0)	47018.2 (0)	531514 (0)	33598.6 (0)	151994.2 (0)	151525.4 (0)
		90	229470.8 (0)	97638.6 (0)	1103596 (5)	72544.4 (0)	275612.6 (6)	235776.7 (9)
		110	195819.1 (10)	157547.9 (5)	681660.1 (10)	73505.6 (10)	71013.9 (10)	72963.3 (10)
4	KP	20	7651.6 (0)	2103.6 (0)	8442.4 (0)	1899.4 (0)	8998 (0)	5043.2 (0)
		30	43529.8 (0)	8469.2 (0)	49641 (0)	7589.4 (0)	53294.2 (0)	26693.4 (0)
		40	330758.7 (2)	82242.8 (2)	430534.1 (2)	72322.6 (1)	340657.3 (7)	221146.5 (3)
	UFLP	42	62617.8 (0)	29925.4 (0)	199163 (0)	22728.8 (0)	68111.2 (0)	68070.2 (0)
		56	71905 (10)	49605.7 (10)	172122.8 (10)	32013.5 (10)	183316.4 (10)	100648.1 (10)

Table 4.1: The average number of nodes explored over 10 instances for each problem class, number of objectives, number of variables, and configuration. The number in brackets is the number of instances unsolved. Note that when the number of unsolved instances is high, the number of nodes explored may be low due to the fact that the algorithm could not explore a large number of nodes within the time limit of one hour.

algorithm benefits the most from probing when tight objective branching constraints (i.e., small sub-problems) are generated.

Finally, it appears from Figure 4.4 that when no objective branching is used (`NOB`), probing (`VF`) can be slightly beneficial (KP) but can also worsen the performance of the algorithm (CFLP, UFLP). This implies that probing is efficient mainly *in combination* with objective branching in MOBB.

The speed-up can be largely explained by looking at the size of the branch-and-bound tree. In Table 4.1, the average number of nodes explored is given. The first observation is that the number of nodes explored is positively correlated to the CPU time. Indeed, for `NOB`, small differences are observed between `NVF` and `VF` in terms of the number of nodes, whereas larger gaps are observed for `CB`. Finally, similarly to the CPU time, the most significant differences between `NVF` and `VF` are observed for `FOB`. Indeed, if we consider all instances for which both `FOB-NVF` and `FOB-VF` are solved, `FOB-VF` resulted in 12.78 times fewer nodes than `FOB-NVF` for $p = 3$. In many cases, `FOB-NVF` is the configuration with the largest tree size, whereas `FOB-VF` has the smallest. This indicates that probing strongly helps to reduce the size of the tree, both by improving lower bound sets and helping the algorithm to make better branching decisions. Note that a similar observation was made by Adelgren and Gupte (2022) for the bi-objective case.

Finally, one may notice that the speed-up in terms of CPU time is smaller than the gain

p	pb	n	`FOB-VF`	`FOB-VFD`
3	CFLP	65	10.67	10.41
		230	2332.70	2277.70
		495	3601.33	3601.21
	KP	40	65.79	64.59
		50	172.35	167.68
		60	749.72	693.92
		70	2201.65	2033.25
		80	3140.04	3048.65
	UFLP	56	197.04	183.46
		72	677.64	619.65
		90	1934.35	1805.70
		110	3600.15	3600.09
4	KP	20	8.80	8.85
		30	53.94	53.68
		40	1137.97	1138.19
	UFLP	42	489.18	486.94
		56	3600.20	3600.14

Table 4.2: Average CPU time expressed in seconds over 10 instances for each problem class, number of objectives, and number of variables. Two configurations are tested: `VF` and `VFD`, both in combination with `FOB`.

in terms of the number of nodes explored. This is due to the fact that probing has a significant cost: on average, 23.5% and 40.36% of the total CPU time for `CB` and `FOB` respectively, over all instances.

4.7.2 Combining probing and bounding

We are now interested in the impact of introducing a weighted-sum of the objectives as an objective function when performing probing, as suggested in Section 4.5.1. It is important to test both cases (`VF` and `VFD`) because on the one hand, introducing an objective function improves the fathoming potential, but, on the other hand, may also result in linear programs becoming more difficult to solve. For this purpose, we now consider only two configurations: `FOB-VF` and `FOB-VFD`. A weight of $(1, ..., 1)$ is used for the weighted-sum of the objectives in `FOB-VFD`.

The performance of the two configurations is given in Table 4.2. On average, it appears that the objective function has a minimal impact on the CPU time. Except in rare cases (e.g. KP for $p = 4$), introducing the objective function is still slightly beneficial. A speed-up of 4.64% is observed on average across all instances for which both `FOB-VF` and `FOB-VFD` are solved.

A potential explanation for the low gain is that the linear relaxation of a weighted-sum scalarization provides a rather weak lower bound set. This is true in particular in the case where $p \geq 3$, as many local upper bounds can have infinite components. Such local upper bounds have an infinite weighted-sum value and are thus always dominated by the lower bound set obtained here. In comparison, at most, two local upper bounds can take infinite components in the bi-objective case. One could imagine a strategy where if a local upper bound with an infinite value on component k is discovered in the sub-problem at hand, a value of 0 is set in component k of the weight vector λ. However, the lower bound set then becomes rather weak, as it is more difficult to reach the local upper bounds that take a large value on the remaining objective. This strategy was tested in preliminary experiments, but similarly to the weight $(1, ..., 1)$, it led to small speed-ups only.

4.7.3 Node selection rules

We now investigate the performance of the node selection rules proposed in Section 4.6. In this section, we use `FOB-VF` as the basic configuration, and combine it with four node selection strategies: `DB`, `BBWS`, `BBWSN`, and `BBGAP`.

The results are given in Figure 4.5. The x-axis represents the time elapsed in seconds, and the y-axis corresponds to the proportion of the instances solved. Hence, each curve represents the proportion of instances solved over time for each configuration. The first observation is that in most cases, `BBWS` performs better than `DB`, and similarly in worst cases. This is, in fact, a promising result because `BBWS` both did not require any fine-tuning of the node selection rule depending on the problem class, and performed better on average. Indeed, as explained earlier, configuration `DB` uses depth-first for KP and breadth-first for CFLP and UFLP, and it is not rare to see either depth-first or breadth-first used in the literature. However, a problem dependent node selection rule is not desirable in a generic solver such as MOBB. Hence, the fact that `BBWS` performs better on average than `DB` overcomes this significant drawback, and increases the efficiency of the MOBB framework as well.

For KP and UFLP, `BBWS` and `BBWSN` perform similarly. This is expected, since all objective coefficients take value in the same range for each objective function for both problem classes. This is, however, not the case for CFLP, and an interesting difference in the performance of `BBWS` and `BBWSN` is observed. In particular, `BBWSN` is faster than `BBWS`, which suggests that it is more beneficial to consider a weight vector that does not favor some of the objectives.

Finally, `BBGAP` is the worst configuration, despite exploring on average 31% fewer nodes than `BBWSN` over all instances solved for both configurations. This is due to the heavy computational cost of computing the gap measures in each node. Indeed, updating the gap measure when selecting a node (Algorithm 4.3) uses, on average, 17% of the total CPU time

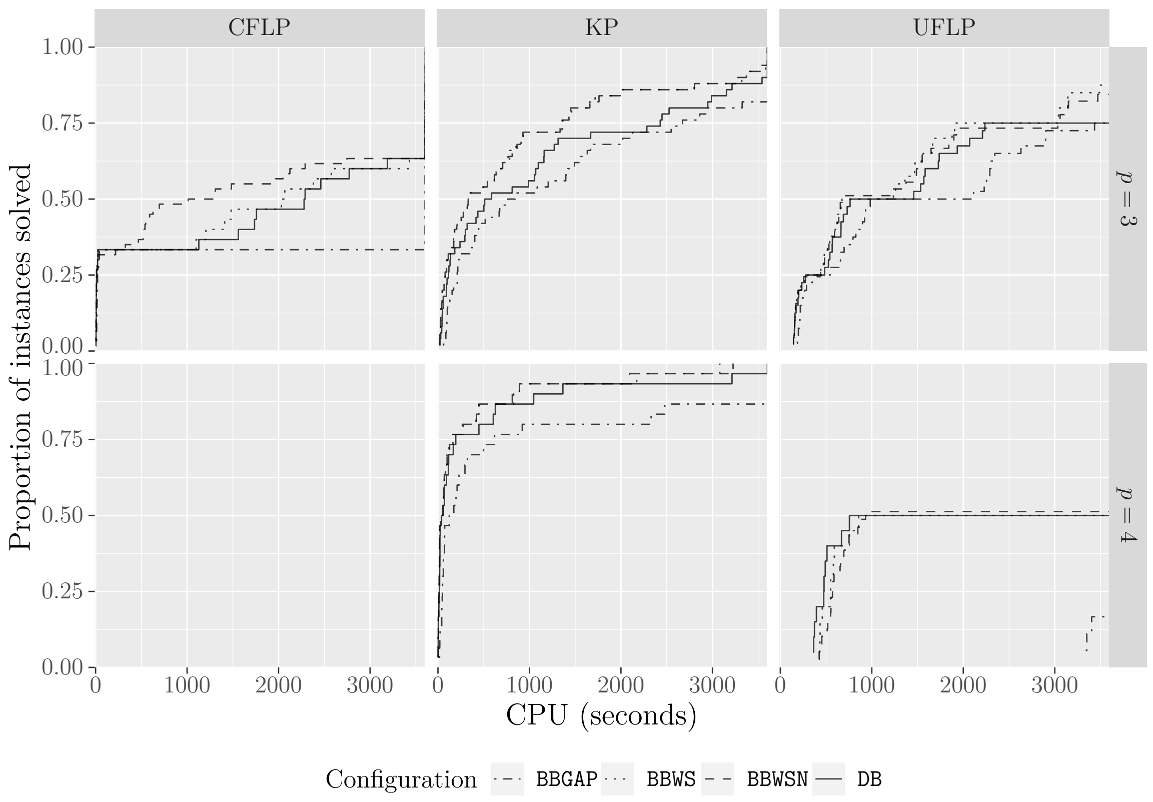


Figure 4.5: Performance curves of the different node selection configurations in combination with `FOB-VF`. The x-axis represents the CPU time expressed in second, and the y-axis corresponds to the proportion of instances solved. Each curve represents the proportion of instances solved through time for a given configuration.

for $p = 3$, and 36% for $p = 4$. Combined with the fact that the gap of each node has to be computed at its creation as well, the total cost of this node selection rule overwhelms the gain obtained from the reduction of the number of nodes, leading to worse CPU times. This, however, suggests that better CPU times could possibly be achieved if a less computationally heavy gap measure is used.

4.7.4 Objective branching and cover cuts inequalities

In this section, we investigate cover cuts inequalities generated from the objective branching constraints. We focus on UFLP, as all of its objectives are in minimization. Let s be the super local upper bound that defines the objective branching constraints, cover cuts are generated at each node for each objective bounded by a constraint, and on each extreme point l of the lower bound set that has non-integer pre-images and that satisfies $l \leqq s$. The cover cuts are generated after objective branching and before variable fixing, constructed following the idea presented in Section 4.5.2, and lifted using the procedure from Letchford and Souli (2019).

		CPU		# nodes		# unsolved		# LP (LP relax)	
p	n	`NCC`	`CC`	`NCC`	`CC`	`NCC`	`CC`	`NCC`	`CC`
3	56	177.31	168.35	12282.80	11275.40	0	0	252691.10	249869.10
	72	564.35	593.53	26574.40	23447.70	0	0	634866.90	648676.70
	90	1590.54	1712.46	52141.60	50846.00	0	0	1395326.90	1487256.30
	110	3381.68	3600	80985.70	45316.30	5	10	2530157.00	2124939.10
4	42	558.93	668.49	22393.80	21351.60	0	0	688906.20	730118.70
	56	3600	3600	39079.10	32976.78	10	10	2536207.70	2612500.70

Table 4.3: Only results for UFLP are shown in this table. For each number of objectives and number of variables, it reports the average cpu time (CPU), the average number of node explored (# nodes), the number of instances unsolved (# unsolved), and the average total number of linear programs solved to compute the linear relaxation (# LP (LP relax)).

Note that it often occurs that no cover cut is generated on an extreme point, for example, if the point is located too far from any of the objective branching constraints, or if the newly generated cut is redundant with an already existing one.

In this section, we consider only the best previous configuration, namely `FOB-VF-BBWSN`. Then, two configurations are tested: with (`CC`) and without cover cuts inequalities (`NCC`).

The results of this experiment are given in Table 4.3. The configuration `CC` generally performs slightly worse than `NCC`. This result can be a surprise when looking at the number of nodes in Table 4.3, as it is generally slightly lower for `CC`. This suggests that the cuts come with a hidden cost that is not compensated enough with the decrease in the size of the tree.

This cost can be understood by looking at the last column of Table 4.3, where the average total number of linear programs solved to compute the lower bound sets in an entire tree is reported. It tends to increase when cover cuts are used, which implies that the lower bound sets are more complex, and require more linear programs to be solved for `CC`. This observation makes sense as new constraints are added to the problem. Furthermore, when computing the lower bound set, these constraints are expected to often be binding as they cut an extreme point from the lower bound set in the father node. This possibly results in more facets to generate and, consequently, more computational effort required. This phenomenon highlights a difficulty of cut generation in the multi-objective case: even simple cuts can become a computational burden, because they can easily complexify the lower bound set, possibly resulting in a huge additional effort to obtain only small improvements.

4.7.5 Investigating variable selection rules: the case of CFLP

By performing probing, the algorithm is able to reduce the set of branching candidates. This indirectly helps the algorithm to make better branching decisions, as variables that

	CPU		# unsolved	
n	`MOF`	`PS`	`MOF`	`PS`
65	9.98	6.59	0	0
230	1988.02	506.50	1	0
495	3600	3600	10	10

Table 4.4: Only results for CFLP are shown in this table. For each number of variables, it reports the average CPU time expressed in seconds (CPU) and the number of unsolved instances (# unsolved) for two configurations considered, namely `MOF` and `PS`.

lead to redundant branches are discarded. In this section, we study whether having a good variable selection rule could help further improve the CPU time. For this purpose, we limit the analysis to CFLP, for which a good branching rule is well known in the single objective case: branch first on the variables that handle the opening of the facilities. We limit ourselves to the best known configuration so far, i.e., `FOB-VFD-BBWSN`. No cover cuts are generated. Two configurations are tested here: the regular variable selection rule (`MOF`, for Most Often Fractional), and the problem specific selection rule (`PS`, for Problem Specific).

The results presented in Table 4.4 clearly show that the problem specific rule (`PS`) systematically outperforms the generic rule (`MOF`). This implies that there is much room for improvement in the design of the variable selection rule. This aspect of MOBB has not received a lot of attention in the literature to this day, and this is left for future research.

4.7.6 Enumeration in MOBB

In the deepest parts of the tree, only a few variables are free. Consequently, the lower bound sets are also expected to be very simple, i.e., made of a few facets and extreme points only. According to Forget et al. (2022), each facet and extreme point possibly requires solving one linear program. Given the fact that the number of free variables is low, one may enumerate all possible solutions and update the upper bound set accordingly instead of keeping branching and computing lower bound sets until all nodes are fathomed. Here, after performing probing, if less than 14 variables are free, all $2^{14} = 16384$ solutions are enumerated. This value has been determined with preliminary experiments on a subset of instances.

Results are reported in Table 4.5. Configuration `BB` corresponds to the best generic branch-and-bound so far, i.e., it uses full objective branching (`FOB`), variable fixing with an objective function (`VFD`), and the best-bound node selection rule based on the normalized weighted-sum idea (`BBWSN`). Configuration `BB-E` uses the same parameters, except that enumeration as described above is enabled.

In general, the enumeration procedure reduces the CPU time, even for larger instances.

p	pb	n	BB	BB-E	OSS
3	CFLP	65	9.98	10.04	**2.31**
		230	1988.02	1993.33	**144.84**
		495	3601.69	3601.48	**731.43**
	KP	40	46.73	42.45	**12.09**
		50	116.50	108.20	**24.83**
		60	457.80	430.96	**73.45**
		70	1414.09	1341.10	**191.77**
		80	2491.45	2388.85	**317.10**
	UFLP	56	177.31	**134.13**	197.66
		72	564.35	**421.29**	545.01
		90	1590.54	**1229.19**	1342.05
		110	3381.68	3132.80	**2964.97**
4	KP	20	6.99	**2.37**	8.59
		30	42.00	**35.99**	36.12
		40	854.79	751.85	**320.28**
	UFLP	42	558.93	**306.41**	686.46
		56	3600.20	**3513.76**	3518.88

Table 4.5: Average CPU time expressed in seconds over 10 instances for each problem class, number of objectives, and number of variables. Three configurations are tested here: BB, BB-E, and OSS. The fastest configuration of the three is in bold.

For CFLP, enumeration appears to be slightly slower (e.g., less than 0.5% for $n = 230$). As expected, the proportion of CPU time spent in updating the upper bound set is greater in BB-E, but fewerlinear programs are solved.

4.7.7 Comparison with objective space search algorithms

In this section, we compare our MOBB framework to an objective space search algorithm. For the comparison, we use a C++ implementation of the redundancy avoidance method from Klamroth et al. (2015). The implementation was kindly shared with us by Dächert et al. (2021). The idea is to decompose the objective space based on the local upper bounds, and to explore each sub-regions independently while avoiding redundant regions. In that aspect, the algorithm we use is similar to Tamby and Vanderpooten (2021), except that the implementation we use does not go as far as theirs in the fine-tuning of CPLEX's parameters.

The OSS algorithm is compared to the best generic configuration of our MOBB, namely BB-E. No cover cuts are generated, and the generic variable selection rule is used for CFLP (MOF). Consequently, two configurations are tested here: our branch-and-bound framework (BB-E) and the objective space search algorithm (OSS). The results are reported in Table 4.5.

First, `BB-E` seems to be competitive with `OSS` for UFLP, meaning that the first step towards an efficient branch-and-bound framework for the multi-objective case has been reached. There is, however, a tendency for larger instances to be solved faster by `OSS`, e.g., for UFLP, $p = 3$, and $n = 110$. This suggests that there are costs that do not occur for small and medium sized instances but that become a burden for larger instances. In our opinion, this constitutes a direction for future research. Indeed, now that the branch-and-bound algorithm is competitive on medium-sized instances, the next step is to tackle larger problems.

Regarding CFLP and KP, `OSS` is significantly faster than `BB-E`. We have seen in Section 4.7.5 that major improvements could be obtained for CFLP by working on variable selection rules in the future. Gadegaard et al. (2019) showed that cut generation at the root node is very effective for bi-objective CFLP, and this may apply to any number of objectives as well as to other problem classes. These two elements constitute other research directions for the future.

Finally, it is interesting to note that `BB-E` is the most efficient on problems for which the number of non-dominated points per variable ($|\mathcal{Y}_N|/n$) is the highest. Indeed, over all instances solved by at least one of the configurations, UFLP has 102.84 non-dominated points per variable, whereas KP and CFLP have 18.74 and 5.13, respectively. The results from this section confirm the intuition that OSS algorithms are more efficient on problems with fewer non-dominated points, and benefit a lot from the decades of progress embedded in single-objective solvers, which are particularly competitive on problems with a high number of variables.

4.8 Conclusion

In this paper, we first enhanced objective branching for MOBB with three or more objective functions. The experiments showed that combining probing and objective branching leads to significantly smaller search trees, resulting in lower CPU times.

Then, we proposed node selection rules based on the best-bound idea. Two variants were considered, depending on how the quality of a bound is measured: either based on the minimal value of a weighted-sum scalarization or on the smallest gap between upper and lower bound sets. The experiments showed that the former is the most efficient, and performs better than both the traditional depth-first and breadth-first strategies from the literature. Besides, we have observed that computing gaps during the resolution can be expensive. This opens the discussion on appropriate gap measures for multi-objective optimization, in particular for $p \geq 3$, and how to efficiently compute these gaps.

Moreover, other developments on additional features were explored. We first tried to generate cuts based on the objective branching constraints, but this resulted in slower performances due to the increased complexity of the lower bound sets. Instead, generating cuts in the root node may be more beneficial, as done by Gadegaard et al. (2019) for the bi-objective case. This is left for future research. Afterward, we showed in the experiments that although variable fixing reduces the set of potential branching candidates when doing decision space branching, there are still possible improvements to be achieved by identifying appropriate variables to branch on. This constitute another direction for future research. Then, enumeration techniques were tested and generated a speed-up in most cases.

Finally, our branch-and-bound framework proved to be competitive against an objective space search algorithm on UFLP. A possible step for future research is to develop techniques for solving large scale problems.

Acknowledgement

This research was funded in whole, or in part, by the Austrian Science Fund (FWF) [P 31366-NBL]. For the purpose of open access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

Bibliography

Achterberg, T., T. Koch, and A. Martin (2005, jan). Branching rules revisited. *Operations Research Letters 33*(1), 42–54.

Adelgren, N. and A. Gupte (2022, mar). Branch-and-bound for biobjective mixed-integer linear programming. *INFORMS Journal on Computing 34*(2), 909–933.

Al-Rabeeah, M., A. Al-Hasani, S. Kumar, and A. Eberhard (2020, mar). Enhancement of the improved recursive method for multi-objective integer programming problem. *Journal of Physics: Conference Series 1490*(1), 012061.

An, D., S. N. Parragh, M. Sinnl, and F. Tricoire (2022). A matheuristic for tri-objective binary integer programming. Preprint.

Aneja, Y. P. and K. P. K. Nair (1979). Bicriteria transportation problem. *Management Science 25*(3), 73–78.

Bektaş, T. (2018, nov). Disjunctive programming for multiobjective discrete optimisation. *INFORMS Journal on Computing 30*(4), 625–633.

Belotti, P., B. Soylu, and M. Wiecek (2016, nov). Fathoming rules for biobjective mixed integer linear programs: Review and extensions. *Discrete Optimization 22*, 341–363.

Belotti, P., B. Soylu, and M. M. Wiecek (2013). A branch-and-bound algorithm for biojbective mixed-intger programs. Technical report, Clemson University.

Benson, H. P. (1998). An outer approximation algorithm for genrating all efficient extreme points in the outcome set of a multiple objective linear programming problem. *Journal of Global Optimization 13*, 1–24.

Bérubé, J.-F., M. Gendreau, and J.-Y. Potvin (2009, apr). An exact e-constraint method for bi-objective combinatorial optimization problems: Application to the traveling salesman problem with profits. *European Journal of Operational Research 194*(1), 39–50.

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

Boland, N., H. Charkhgard, and M. Savelsbergh (2016). The l-shape search method for triobjective integer programming. *Mathematical Programming Computation 8*(2), 217–251.

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

Boland, N. and H. C. M. Savelsbergh (2016, Jun). The l-shape search method for triobjective integer programming. *Mathematical Programming Computation 8*(2), 217–251.

Byrd, R. H., A. J. Goldman, and M. Heller (1987). Technical note—recognizing unbounded integer programs. *Operations Research 35*(1), 140–142.

Chalmet, L., L. Lemonidis, and D. Elzinga (1986, may). An algorithm for the bi-criterion integer programming problem. *European Journal of Operational Research 25*(2), 292–300.

Clímaco, J. C. N. and M. M. B. Pascoal (2016, sep). An approach to determine unsupported non-dominated solutions in bicriteria integer linear programs. *INFOR: Information Systems and Operational Research 54*(4), 317–343.

Clímaco, J. C. N. and M. M. B. Pascoal (2016). An approach to determine unsupported non-dominated solutions in bicriteria integer linear programs. *INFOR: Information Systems and Operational Research 54*(4), 317–343.

Csirmaz, L. (2015, jun). Using multiobjective optimization to map the entropy region. *Computational Optimization and Applications 63*(1), 45–67.

Dächert, K., T. Fleuren, and K. Klamroth (2021). A simple, efficient and versatile objective space algorithm for multiobjective integer programming. working paper.

Dächert, K. and K. Klamroth (2015, Apr). A linear bound on the number of scalarizations needed to solve discrete tricriteria optimization problems. *Journal of Global Optimization 61*(4), 643–676.

Dächert, K., J. Gorski, and K. Klamroth (2012, dec). An augmented weighted tchebycheff method with adaptively chosen parameters for discrete bicriteria optimization problems. *Computers & Operations Research 39*(12), 2929–2943.

Dächert, K. and K. Klamroth (2015). A linear bound on the number of scalarizations needed to solve discrete tricriteria optimization problems. *Journal of Global Optimization 61*(4), 643–676.

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

Ehrgott, M. (2005). *Multicriteria Optimization* (2nd ed.). Springer Berlin, Heidelberg.

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

Filho, A. A., A. C. Moretti, M. V. Pato, and W. A. de Oliveira (2019, jul). An exact scalarization method with multiple reference points for bi-objective integer linear optimization problems. *Annals of Operations Research 296*(1-2), 35–69.

Florios, K., G. Mavrotas, and D. Diakoulaki (2010). Solving multiobjective, multiconstraint knapsack problems using mathematical programming and evolutionary algorithms. *European Journal of Operational Research 203*(1), 14 – 21.

Forget, N. (2021). C++ implementation of multi-objective branch-and-bound. `https://github.com/NicolasJForget/LinearRelaxationBasedMultiObjectiveBranchAndBound`.

Forget, N., S. L. Gadegaard, K. Klamroth, L. R. Nielsen, and A. Przybylski (2022, aug). Branch-and-bound and objective branching with three or more objectives. *Computers & Operations Research.*

Forget, N., S. L. Gadegaard, and L. R. Nielsen (2022, nov). Warm-starting lower bound set computations for branch-and-bound algorithms for multi objective integer linear programs. *European Journal of Operational Research 302*(3), 909–924.

Forget, N., L. Nielsen, and S. Gadegaard (2020a). Computational results (all instances). Technical report, Aarhus University. Results for all the instances at the repository MOrepo-Forget20.

Forget, N., L. R. Nielsen, and S. L. Gadegaard (2020b). Examples of lower and upper bound sets (morepo-forget20).

Forget, N., L. R. Nielsen, and S. L. Gadegaard (2020c). Instances and results for linear relaxation based branch-and-bound (morepo-forget21). `https://github.com/MCDMSociety/MOrepo-Forget21`.

Forget, N., L. R. Nielsen, and S. L. Gadegaard (2020d). Instances and results for linear relaxation based branch-and-bound (morepo-kirlik14).

Fukuda, K. and A. Prodon (1996). Double description method revisited. In *Lecture Notes in Computer Science*, Volume 1120, pp. 91–111. Berlin: Springer.

Gadegaard, S. (2016). *Discrete Location Problems–Theory, Algorithms, and Extensions to Multiple Objectives*. Ph. D. thesis, Department of Economics and Business Economics, Aarhus BSS, Aarhus University.

Gadegaard, S., L. Nielsen, and M. Ehrgott (2019). Bi-objective branch-and-cut algorithms based on lp relaxation and bound sets. *INFORMS Journal on Computing 31*(4), 790–804.

Goeffrion, A. (1968). Proper efficiency and the theory of vector maximization. *Journal of Mathematical Analysis and Applications 22*(3), 618–630.

Gu, Z., G. L. Nemhauser, and M. W. Savelsbergh (1998). Lifted cover inequalities for 0-1 integer programs: Computation. *INFORMS Journal on Computing 10*(4), 427–437.

Haimes, Y. Y., L. S. Lasdon, and D. A. Wismer (1971). On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems Man and Cybernetics 1*(1), 296–297.

Hamacher, H. W., C. R. Pedersen, and S. Ruzika (2007, may). Finding representative systems for discrete bicriterion optimization problems. *Operations Research Letters 35*(3), 336–344.

Hamel, A. H., A. Löhne, and B. Rudloff (2013, aug). Benson type algorithms for linear vector optimization and applications. *Journal of Global Optimization 59*(4), 811–836.

Holzmann, T. and J. Smith (2018, dec). Solving discrete multi-objective optimization problems using modified augmented weighted tchebychev scalarizations. *European Journal of Operational Research 271*(2), 436–449.

Jozefowiez, N., G. Laporte, and F. Semet (2012, nov). A generic branch-and-cut algorithm for multiobjective optimization problems: Application to the multilabel traveling salesman problem. *INFORMS Journal on Computing 24*(4), 554–564.

Kirlik, G. (2014). Test instances for multiobjective discrete optimization problems.

Kirlik, G. and S. Sayın (2014, aug). Computing the nadir point for multiobjective discrete optimization problems. *Journal of Global Optimization 62*(1), 79–99.

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

Kiziltan, G. and E. Yucaoğlu (1983, December). An algorithm for multiobjective zero-one linear programming. *Management Science 29*(12), 1444–1453.

Klamroth, K., R. Lacour, and D. Vanderpooten (2015). On the representation of the search region in multi-objective optimization. *European Journal of Operational Research 245*(3), 767–778.

Klein, D. and E. Hannan (1982). An algorithm for the multiple objective integer linear programming problem. *European Journal of Operational Research 9*(4), 378 – 385.

Leitner, M., I. Ljubić, M. Sinnl, and A. Werner (2016, aug). ILP heuristics and a new exact method for bi-objective 0/1 ILPs: Application to FTTx-network design. *Computers & Operations Research 72*, 128–146.

Lemesre, J., C. Dhaenens, and E. Talbi (2007, aug). Parallel partitioning method (PPM): A new exact method to solve bi-objective problems. *Computers & Operations Research 34*(8), 2450–2462.

Letchford, A. N. and G. Souli (2019, mar). On lifted cover inequalities: A new lifting procedure with unusual properties. *Operations Research Letters 47*(2), 83–87.

Linderoth, J. T. and M. W. Savelsbergh (1999). A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing 11*(2), 173–187.

Löhne, A. and B. Weißing (2020). Bensolve - vlp solver, version 2.1.x. `http://www.bensolve.org`.

Lokman, B. and M. Köksalan (2012, jul). Finding all nondominated points of multi-objective integer programs. *Journal of Global Optimization 57*(2), 347–365.

Mavrotas, G. (2009, jul). Effective implementation of the e-constraint method in multi-objective mathematical programming problems. *Applied Mathematics and Computation 213*(2), 455–465.

Mavrotas, G. and D. Diakoulaki (1998). A branch and bound algorithm for mixed zero-one multiple objective linear programming. *European Journal of Operational Research 107*(3), 530–541.

Mavrotas, G. and D. Diakoulaki (2005). Multi-criteria branch and bound: A vector maximization algorithm for mixed 0-1 multiple objective linear programming. *Applied Mathematics and Computation 171*(1), 53–71.

Mavrotas, G. and K. Florios (2013, may). An improved version of the augmented e-constraint method (AUGMECON2) for finding the exact pareto set in multi-objective integer programming problems. *Applied Mathematics and Computation 219*(18), 9652–9669.

Nemhauser, G. and L. Wolsey (1999). *Integer and Combinatorial Optimization.* John Wiley & Sons.

Nielsen, L. (2020). *gMOIP: Tools for 2D and 3D Plots of Single and Multi-Objective Linear/Integer Programming Models.* v1.4.3.

Ozlen, M., B. Burton, and C. MacRae (2014, Feb). Multi-objective integer programming: An improved recursive algorithm. *Journal of Optimization Theory and Applications 160*(2), 470–482.

Parragh, S. and F. Tricoire (2019). Branch-and-bound for bi-objective integer programming. *INFORMS Journal on Computing 31*(4), 805–822.

Pettersson, W. and M. Ozlen (2017, oct). A parallel approach to bi-objective integer programming. *ANZIAM Journal 58*, 69.

Pettersson, W. and M. Ozlen (2019, sep). Multiobjective integer programming: Synergistic parallel approaches. *INFORMS Journal on Computing 32*(2), 461–472.

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

Przybylski, A., X. Gandibleux, and M. Ehrgott (2010). 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.

Ralphs, T. K., M. J. Saltzman, and M. M. Wiecek (2006, sep). An improved algorithm for solving biobjective integer programs. *Annals of Operations Research 147*(1), 43–70.

Ramos, R. M., S. Alonso, J. Sicilia, and C. González (1998). The problem of the optimal biobjective spanning tree. *European Journal of Operational Research 111*(3), 617 – 628.

Santis, M. D., G. Eichfelder, J. Niebling, and S. Rocktäschel (2020, jan). Solving multiobjective mixed integer convex optimization problems. *SIAM Journal on Optimization 30*(4), 3122–3145.

Santis, M. D., G. Grani, and L. Palagi (2020, may). Branching with hyperplanes in the criterion space: The frontier partitioner algorithm for biobjective integer programming. *European Journal of Operational Research 283*(1), 57–69.

Savelsbergh, M. W. (1994). Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing 6*(4), 445–454.

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

Sourd, F. and O. Spanjaard (2008). A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem. *INFORMS Journal on Computing 20*(3), 472–484.

Stidsen, T. and K. A. Andersen (2018). A hybrid approach for biobjective optimization. *Discrete Optimization 28*, 89–114.

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

Sylva, J. and A. Crema (2004). A method for finding the set of non-dominated vectors for multiple objective integer linear programs. *European Journal of Operational Research 158*(1), 46 – 55.

Sylva, J. and A. Crema (2008, jul). Enumerating the set of non-dominated vectors in multiple objective integer linear programming. *RAIRO - Operations Research 42*(3), 371–387.

Tamby, S. and D. Vanderpooten (2021). Enumeration of the nondominated set of multiobjective discrete optimization problems. *INFORMS Journal on Computing 33*(1), 72–85.

Ulungu, E. and J. Teghem (1995). The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences 20*(2), 149–165.

Ulungu, E. L. and J. Teghem (1997). Solving multi-objective knapsack problem by a branch-and-bound procedure. In J. Clímaco (Ed.), *Multicriteria Analysis*, pp. 269–278. Springer Berlin Heidelberg.

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

Visée, M., J. Teghem, M. Pirlot, and E. Ulungu (1998). Two-phases method and branch and bound procedures to solve the bi–objective knapsack problem. *Journal of Global Optimization 12*, 139–155.

Zhang, W. and M. Reimann (2014, apr). A simple augmented e-constraint method for multi-objective mathematical integer programming problems. *European Journal of Operational Research 234*(1), 15–24.

Özlen, M. and M. Azizoğlu (2009, nov). Multi-objective integer programming: A general approach for generating all non-dominated solutions. *European Journal of Operational Research 199*(1), 25–35.

AARHUS BSS SCHOOL OF BUSINESS AND SOCIAL SCIENCES
AARHUS UNIVERSITY

Declaration of co-authorship

Full name of the PhD student: Nicolas Forget

This declaration concerns the following article/manuscript:

Title:	Warm-starting lower bound set computation for branch-and-bound algorithms for multi-objective linear programs
Authors:	Nicolas Forget, Sune Lauth Gadegaard, Lars Relund Nielsen

The article/manuscript is: Published ☒ Accepted ☐ Submitted ☐ In preparation ☐

If published, state full reference: Forget, N., Gadegaard, S. L., and Nielsen, L. R. (2022). Warm-starting lower bound set computation for branch-and-bound algorithms for multi-objective linear programs. *European Journal of Operational Research*, 302(3):909-924.

If accepted or submitted, state journal:

Has the article/manuscript previously been used in other PhD or doctoral dissertations?

No ☒ Yes ☐ If yes, give details:

The PhD student has contributed to the elements of this article/manuscript as follows:

A.	Has essentially done all the work
B.	Major contribution
C.	Equal contribution
D.	Minor contribution
E.	Not relevant

Element	Extent (A-E)
1. Formulation/identification of the scientific problem	A
2. Planning of the experiments/methodology design and development	B
3. Involvement in the experimental work/clinical studies/data collection	B
4. Interpretation of the results	B
5. Writing of the first draft of the manuscript	A
6. Finalization of the manuscript and submission	B

Signatures of the co-authors

Date	Name	Signature
12/8-22	Sune Lauth Gadegaard	[illegible]
4/8/22	Lars Relund Nielsen	Lars Relund

In case of further co-authors please attach appendix

Date: 25/08/2022 [illegible]

Signature of the PhD student

AARHUS BSS SCHOOL OF BUSINESS AND SOCIAL SCIENCES
AARHUS UNIVERSITY

Declaration of co-authorship

Full name of the PhD student: Nicolas Forget

This declaration concerns the following article/manuscript:

Title:	Branch-and-bound and objective branching with three or more objectives
Authors:	Nicolas Forget, Sune Lauth Gadegaard, Kathrin Klamroth, Lars Relund Nielsen, Anthony Przybylski

The article/manuscript is: Published ☐ [✓ Accepted] Submitted ☐ In preparation ☐

If published, state full reference:

If accepted or submitted, state journal: Computers & Operations Research

Has the article/manuscript previously been used in other PhD or doctoral dissertations?

[✓ No] Yes ☐ If yes, give details:

The PhD student has contributed to the elements of this article/manuscript as follows:

- A. Has essentially done all the work
- B. Major contribution
- C. Equal contribution
- D. Minor contribution
- E. Not relevant

Element	Extent (A-E)
1. Formulation/identification of the scientific problem	B
2. Planning of the experiments/methodology design and development	B
3. Involvement in the experimental work/clinical studies/data collection	A
4. Interpretation of the results	B
5. Writing of the first draft of the manuscript	B
6. Finalization of the manuscript and submission	B

Signatures of the co-authors

Date	Name	Signature
12/8-22	Sune Lauth Gadegaard	[illegible]
	Kathin Klamroth	[illegible]
	Lars Relund Nielsen	Lars Relund
	Anthony Przybylski	AP

In case of further co-authors please attach appendix

Date: 25/08/2022

[illegible]

Signature of the PhD student

AARHUS BSS SCHOOL OF BUSINESS AND SOCIAL SCIENCES
AARHUS UNIVERSITY

Declaration of co-authorship

Full name of the PhD student: Nicolas Forget

This declaration concerns the following article/manuscript:

Title:	Enhancing branch-and-bound for multi-objective 0-1 programming
Authors:	Nicolas Forget, Sophie Parragh

The article/manuscript is: Published ☐ Accepted ☐ Submitted ☐ In preparation ☒

If published, state full reference:

If accepted or submitted, state journal: INFORMS Journal on Computing

Has the article/manuscript previously been used in other PhD or doctoral dissertations?

No ☒ Yes ☐ If yes, give details:

The PhD student has contributed to the elements of this article/manuscript as follows:

A. Has essentially done all the work
B. Major contribution
C. Equal contribution
D. Minor contribution
E. Not relevant

Element	Extent (A-E)
1. Formulation/identification of the scientific problem	B
2. Planning of the experiments/methodology design and development	B
3. Involvement in the experimental work/clinical studies/data collection	A
4. Interpretation of the results	B
5. Writing of the first draft of the manuscript	A
6. Finalization of the manuscript and submission	B

Signatures of the co-authors

Date	Name	Signature
	Sophie Parragh	[illegible]

In case of further co-authors please attach appendix

Date: 30/08/2022 [illegible]

Signature of the PhD student