Branch-and-bound and objective branching with three or more objectives[*]

Nicolas Forget[†], Sune Lauth Gadegaard, Lars Relund Nielsen
Department of Economics and Business Economics, School of Business and Social Sciences, Aarhus University, Fuglesangs Allé 4, DK-8210 Aarhus V, Denmark

Kathrin Klamroth
School of Mathematics and Natural Sciences, University of Wuppertal, Gaußstraße 20, 42119 Wuppertal - Germany.

Anthony Przybylski
Faculty of Science and Technology, University of Nantes, 2 Rue de la Houssinière BP 92208, 44322 Nantes Cedex 03 - France.

June 2021

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 lexicographic order among non-dominated outcome 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 is reported.

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

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 optimization* (*MOIP*) problems, where all

[*]Preprint of N. Forget, S.L. Gadegaard, K. Klamroth, L.R. Nielsen and A. Przybylski, *Branch-and-bound and objective branching with three or more objectives* in Computers and Operations Research 2022, doi:10.1016/j.cor.2022.106012

[†]Corresponding author (`nforget@econ.au.dk`).

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 MOIP, called *multi-objective combinatorial optimization* (*MOCO*) problem, has received a specific attention in the literature. Also, although most research has focused on bi-objective MOIP problems, the interest in MOIP problems with more objectives has risen during the last 10-20 years.

The methodology for solving multi-objective optimization problems 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 MOIP 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 MOIP problems was proposed by Sylva and Crema (2004). They solve a series of IPs that becomes more and more constrained as non-dominated outcomes 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 outcome 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 outcomes.

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, Burton, and MacRae (2014) for an algorithm based on recursion that can handle an arbitrary number of objectives; to Dächert and Klamroth (2015) and Klamroth, Lacour, and Vanderpooten (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, Charkhgard, and Savelsbergh (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 outcomes 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 Ramos, Alonso, Sicilia, and González (1998); Ulungu and Teghem (1997, 1995), and Visée, Teghem, Pirlot, and Ulungu (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, Mavrotas, and Diakoulaki (2010). It was later shown by Vincent (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, Seipp, Ruzika, Przybylski, and Gandibleux (2013). The mixed-integer case was also explored by Belotti, Soylu, and Wiecek (2013) and Adelgren and Gupte (2021) 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 pruning 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 (2021) 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 outcomes in the subproblem from the incumbent set.

This idea is further developed in Stidsen, Andersen, and Dammann (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, Nielsen, and Ehrgott (2019) who, independently, propose to partition the outcome space into disjoint areas defined by local nadir points dominated by a lower bound set. Parragh and Tricoire propose to generate the extreme supported outcomes of each node and use those outcomes to generate a lower bound set, whereas Gadegaard et al. use the efficient outcomes 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, Eichfelder, Niebling, and Rocktäschel (2020) and Forget, Gadegaard, and Nielsen (2021) 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. In Santis et al. (2020), they 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 not 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. (2021), 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, Löhne, and Rudloff (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 contributions 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 2. Our multi-objective B&B framework is described in Section 3. Section 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 5, and a conclusion as well as proposals for further research are given in Section 6.

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 $y^1_k \leq y^2_k, \forall k \; \{1, ..., p\}$. 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 $\mathcal{S}_N = \{s \in \mathcal{S} \mid \nexists s' \in \mathcal{S}, \; s' \leqslant s\}$.

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 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 $\mathcal{S}_N \subset cl[\mathbb{R}^p \backslash (\mathcal{U} + \mathbb{R}^p_{\geqq})]$ 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'*.

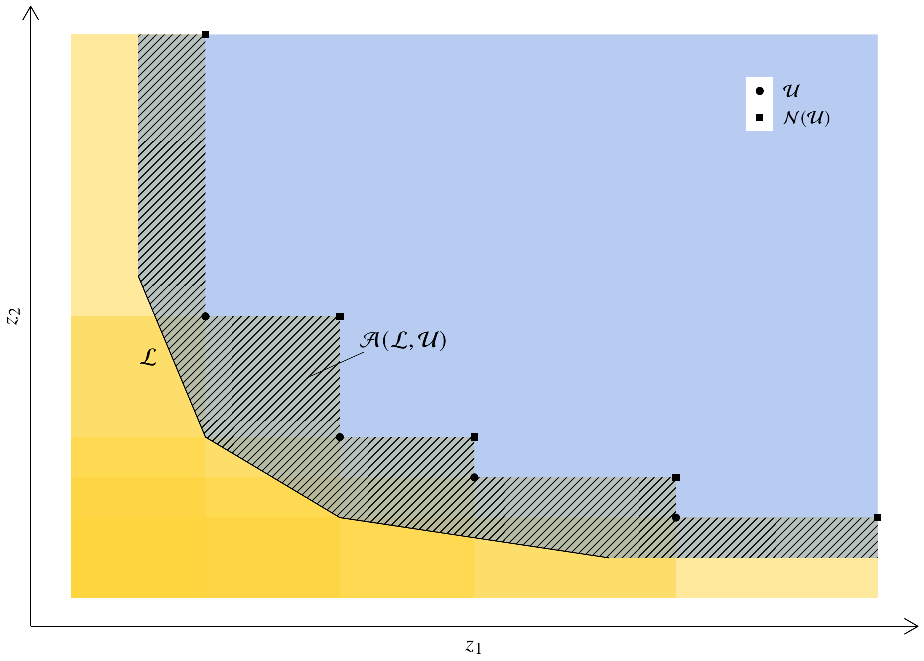
2.2 Search region and local upper bounds

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) 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 1, $\mathcal{Y}_N \subset \mathcal{A}(\mathcal{U}) := \text{cl}(\mathbb{R}^p \backslash (\mathcal{U} + \mathbb{R}^p_{\geqq}))$ (closure of the hatched and yellow areas in Figure 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 2. *Let $C(u) = u - \mathbb{R}^p_{\geqq}$ be the* search cone *given the point $u \in \mathbb{R}^p$. The set of local upper bounds with respect to $\mathcal{U}$, $N(\mathcal{U})$ is a unique discrete set of points in $\mathbb{R}^p$ satisfying*

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



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

Figure 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.

From Definition 2 it follows that $\mathcal{Y}_N \subset \mathcal{A}(\mathcal{U}) = \mathcal{N}(\mathcal{U}) - \mathbb{R}^p_{\geqq}$ and due to Definition 1 we have $\mathcal{Y}_N \subset \mathcal{L} + \mathbb{R}^p_{\geqq}$ (hatched and blue areas in Figure 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 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 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. (2021), 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

Algorithm 1 Multi-objective B&B algorithm

```
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 $\mathcal{Y}_N$.
```

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 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. (2021) 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. (2021) 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 4, we show that it is necessary to compute the dominance test for each local upper bound, i.e., we cannot stop when we find one local upper bound dominated by the lower bound set. Furthermore, in Section 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. (2021).

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. (2021) will be used, and it is recalled in Proposition 1 and Lemma 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 $u' - e$, where $e = (1, ..., 1) \in \mathbb{R}^p$.

Proposition 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 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 1.

Lemma 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 $u \in \mathcal{N}(\mathcal{U})$, there exists $i \in \{1, ..., F\}$ such that $n^{iT} \cdot u < d^i$.*

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 4, and its impact on a multi-objective B&B algorithm is discussed in Section 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 $v \in \mathbb{N}$ (Step 5). Then, either constraint $x_i \leq v$ or constraint $x_i \geq v + 1$ 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 $\mathcal{X}(\eta_0) = \{x \in \mathcal{X}(\eta) \mid x_i \leq v\}$ and $\mathcal{X}(\eta_1) = \{x \in \mathcal{X}(\eta) \mid x_i \geq v + 1\}$. 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 5.

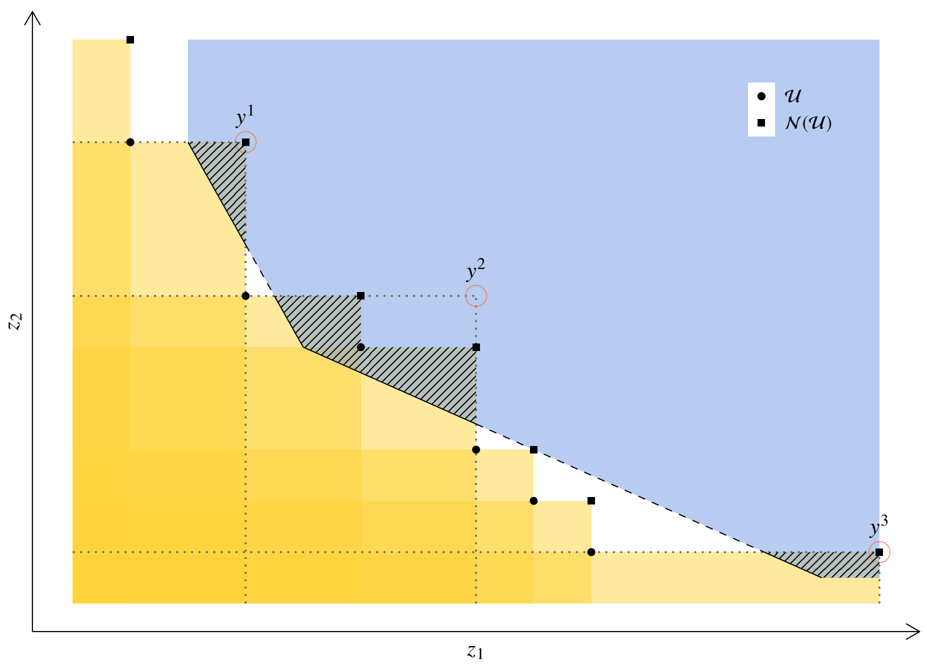


Figure 2: A lower bound set (solid and dashed lines) partially dominated by the upper bound set. The dominated area of the lower bound set is represented by the dashed lines. Three disjoint subproblems are created in the objective space by applying objective branching on y^1, y^2 and y^3, represented 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).

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 and Andersen, 2018; Stidsen et al., 2014). 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.

Definition 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 $P(\eta, \bar{y}) := \min\{Cx \mid x \in \mathcal{X}(\eta),\ z(x) \leqq \bar{y}\}$ 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.

Now consider the situation depicted in Figure 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 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 $P(\eta, y^3)$ respectively. The corresponding constraints are depicted with dotted lines in Figure 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 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:*

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

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

1c) tightness: as much dominated area as possible is discarded

Property 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. Hence, this is a necessary condition, and it cannot be relaxed.

Property 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 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.

Property 1a and Property 1b are the main focus in the paper, but Property 1c will be briefly discussed in the rest of this section and in Section 5.

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 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 1 by applying objective branching on its local upper bounds.

Exactly the same situation is depicted with three objectives in Figure 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 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, in Figure 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 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 4a, it can be observed that there are two non-dominated connected components. However, as seen in Figure 4b, 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.

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.

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 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 $C(u)$ 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

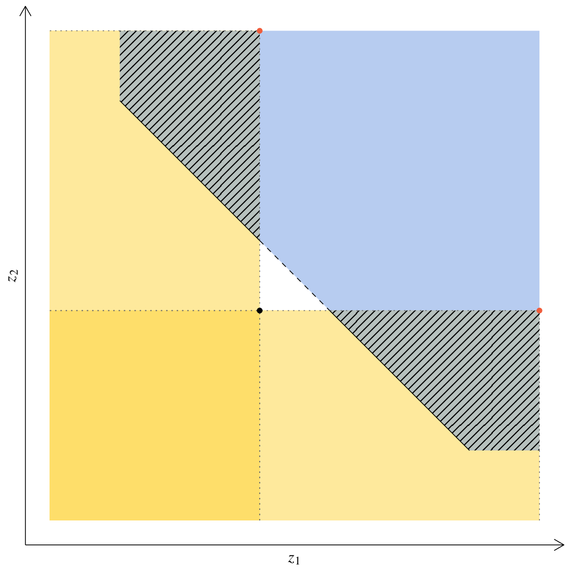
$$\mathcal{D}(\eta) = \{u \in \mathcal{N}(U) \mid \exists\, l \in \mathcal{L}(\eta),\ l \leqslant u\}$$

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.

4.2.1 Merging operations on local upper bounds

As explained in Section 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 4, will be used.

Definition 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 $C(s^I)$ will be called* the intersection cone.



(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 1.

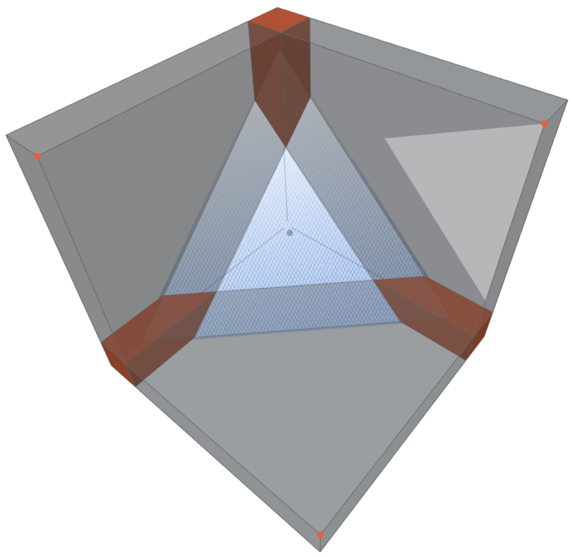

(b) Three-objective case: The dominated part of the lower bound set is the blue area in the middle (view from above). When objective branching is applied on each local upper bound, there are redundancies between the subproblems (each subproblem defines a grey search area). Every point in the brown areas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

Figure 3: 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 3b can be seen in Forget et al. (2020a).

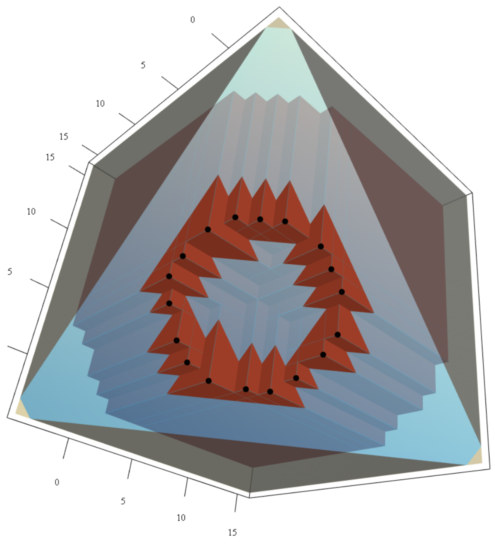
In particular, $C(s^I) = C(s^1) \cap C(s^2)$. Hence, the intersection cone contains all the points of the objective space that are contained in both $P(\eta, s^1)$ and $P(\eta, s^2)$. Thus, if the intersection point s^I is dominated by $\mathcal{L}(\eta)$, there may exist feasible points that are in $C(s^I)$, 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 5. They can be seen as merged local upper bounds.

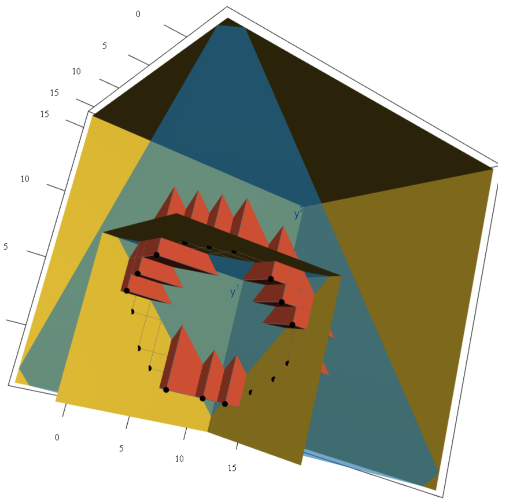
Definition 5. *Consider a set of local upper bounds $u^1, \ldots, u^h \in \mathcal{D}(\eta)$, with $h \in \mathbb{N}\setminus\{0\}$. The point $s \in \mathbb{R}^p$ is a super local upper bound of the local upper bounds $u^1, \ldots, u^h$ if $s_k = \max_{i\in\{1,\ldots,h\}} u^i_k$, $\forall k \in \{1, \ldots, p\}$. Furthermore, if a local upper bound $u \in \mathcal{D}(\eta)$ satisfies $u \subset C(s)$, 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 $\{u^1, \ldots, u^h\}$ 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



(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 4: An example of applying objective branching on two disjoint search areas (view from below). Interactive plots of Figures 4a and 4b can be seen in Forget et al. (2020a)

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.

4.2.2 Desirable properties of the set of super local upper bounds

At each node η, a set of super local upper bounds $\mathcal{S}$ 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 1, a set of desirable properties for $\mathcal{S}$ will be defined here.

Property 2. *The desirable properties of a set of super local upper bounds are the following:*

2a) $\forall u \in \mathcal{D}(\eta)$, $\exists s \in \mathcal{S}$ such that u is contained in s, i.e., $u \in \mathcal{D}(\eta, s)$.

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 $\mathcal{L}(\eta)$.

2c) $\forall s \in \mathcal{S}$, $\forall \epsilon \geqslant 0$, $\exists u \in \mathcal{D}(\eta, s)$ such that $u \notin \mathcal{D}(\eta, s - \epsilon)$.

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 1a, Property 1b, and Property 1c.

Property 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

Algorithm 2 Computation of the set of super local upper bounds

```
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:      $s \leftarrow$ 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
```

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 1a is satisfied.

Property 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 1b is satisfied.

Property 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 1c is satisfied.

Property 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 $C(s)$) 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 verified to ensure, once again, that as much of the dominated region as possible is discarded, and that it maintains Property 1c.

Consequently, obtaining a set $\mathcal{S}$ of super local upper bounds that satisfies Property 2 ensures that as much dominated region as possible is discarded while still satisfying the conditions established in Property 1.

4.2.3 An algorithm to compute a set of super local upper bounds

In this paragraph we describe the algorithm used to compute the set of super local upper bounds $\mathcal{S}$ 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 2. The function `Merge`(s^1, s^2) simply merges s^1 and s^2 as presented in Section 4.2.1.

Theorem 1. *Algorithm 2 computes the set of super local upper bounds.*

Proof. It will be shown that the output $\mathcal{S}$ of this algorithm satisfies Property 2.

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 2a is satisfied.

Algorithm 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 2b is satisfied.

A super local upper bound can be defined as a nadir point of some dominated local upper bounds (Definition 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 2c is satisfied.

In order to violate the Property 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 2, we establish the complexity of Algorithm 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 2. *Algorithm 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 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 2 consequently runs in $O(|\mathcal{D}(\eta)|^3 \cdot f(\mathcal{L}(\eta)))$, and this bound is not tight either. □

4.2.4 Implications of Property 2

Let $\mathcal{S}$ denote a set of super local upper bounds satisfying Property 2.

Proposition 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 2b. Furthermore, by Definition 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 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 $\mathcal{D}(\eta, s^2) = \mathcal{D}(\eta, s) \backslash \{u\}$. The super local upper bounds s^1 and s^2 satisfy Property 2a, Property 2b and Property 2c, and therefore Property 2d is not satisfied. □

Lemma 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 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 2b is not satisfied. □

Lemma 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 2 and in particular Property 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 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 2b. If this is not the case, the same principle applies to u' and u''. As both Lemma 2, Lemma 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 2b.

□

Theorem 3. *The set $\mathcal{S}$ is unique.*

Proof. If $|\mathcal{S}| = 1$, then by Lemma 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 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 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 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 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 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 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 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 2b. The same reasoning can be applied to v' and $\mathcal{S}$. $\square$

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 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 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 way. 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 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.

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, and for the LP solver, the default parameters were used.

When selecting a node to explore (Step 1 of Algorithm 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. (2021) was used for the computation of the linear relaxation at each node (Step 2 of Algorithm 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. (2021) 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 reviled 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 pre-images were the these extreme points 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 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 $C(s)$ will be considered. If multiple choices are possible or if no extreme point is located in $C(s)$, 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 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 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 4.3). This is referred to 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 5.2)?
- Why does objective branching perform the way it does (Section 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 5.4)?

Table 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.

- How does the B&B algorithm perform compared to an objective space search algorithm (Section 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.

5.1 Test instances

A total of 600 instances (see Table 1) taken from the multi-objective literature have been used. Four problem classes are considered: Assignement Problems (AP) from Bektaş (2018), randomly generated Integer Linear Programs (ILP) and Knapsack Problems (KP) from Kirlik (2014) (online at Forget, Nielsen, and Gadegaard (2020c)), Uncapacitated Facility Location Problems (UFLP), and Production Planning Problems (PPP) (online at Forget, Nielsen, and Gadegaard (2020b)). 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 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

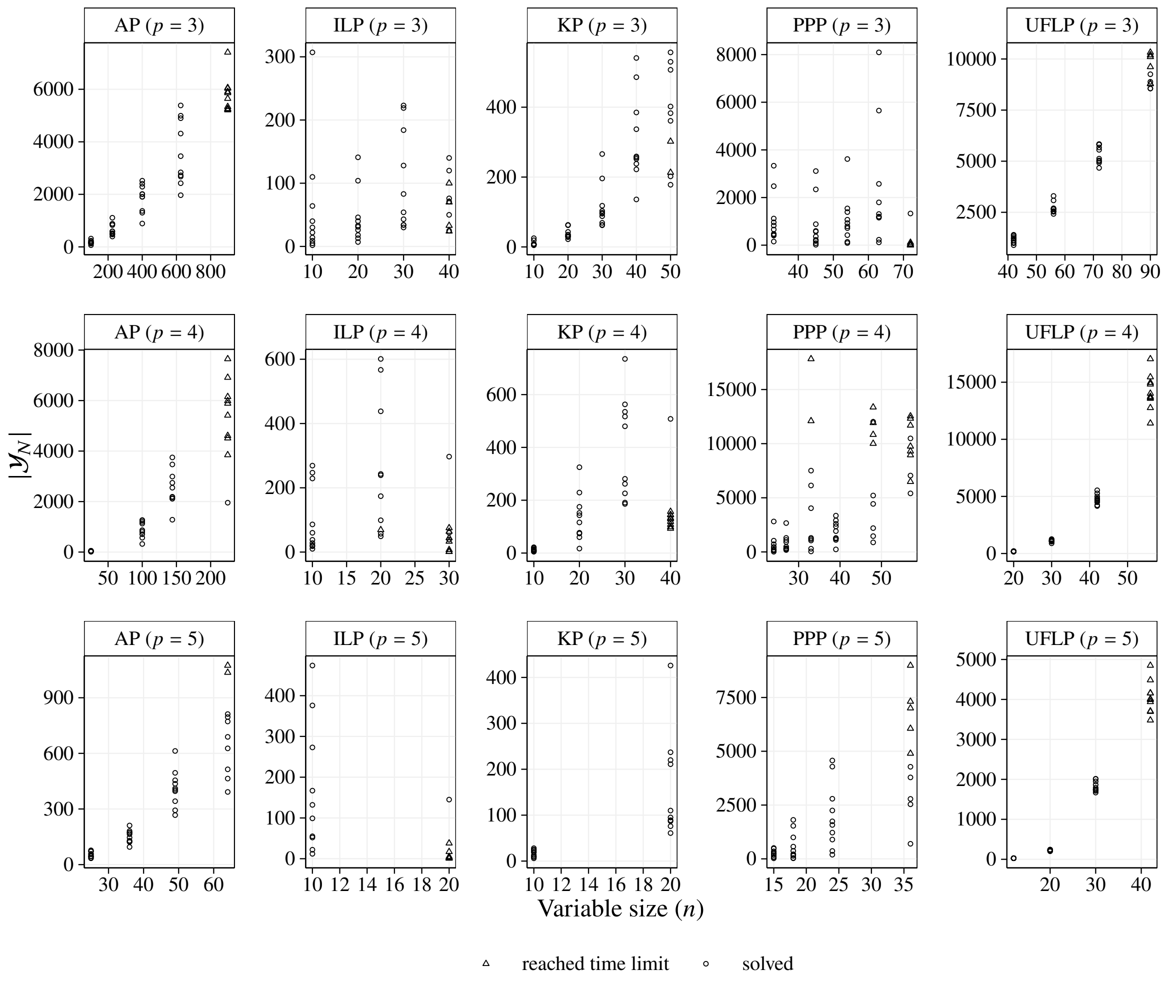


Figure 5: Number of non-dominated points. One point for each instance is given. Instances that have not been solved to optimality are illustrated with a different shape. Note that the scale for each sub-plot is different.

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 in 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 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.

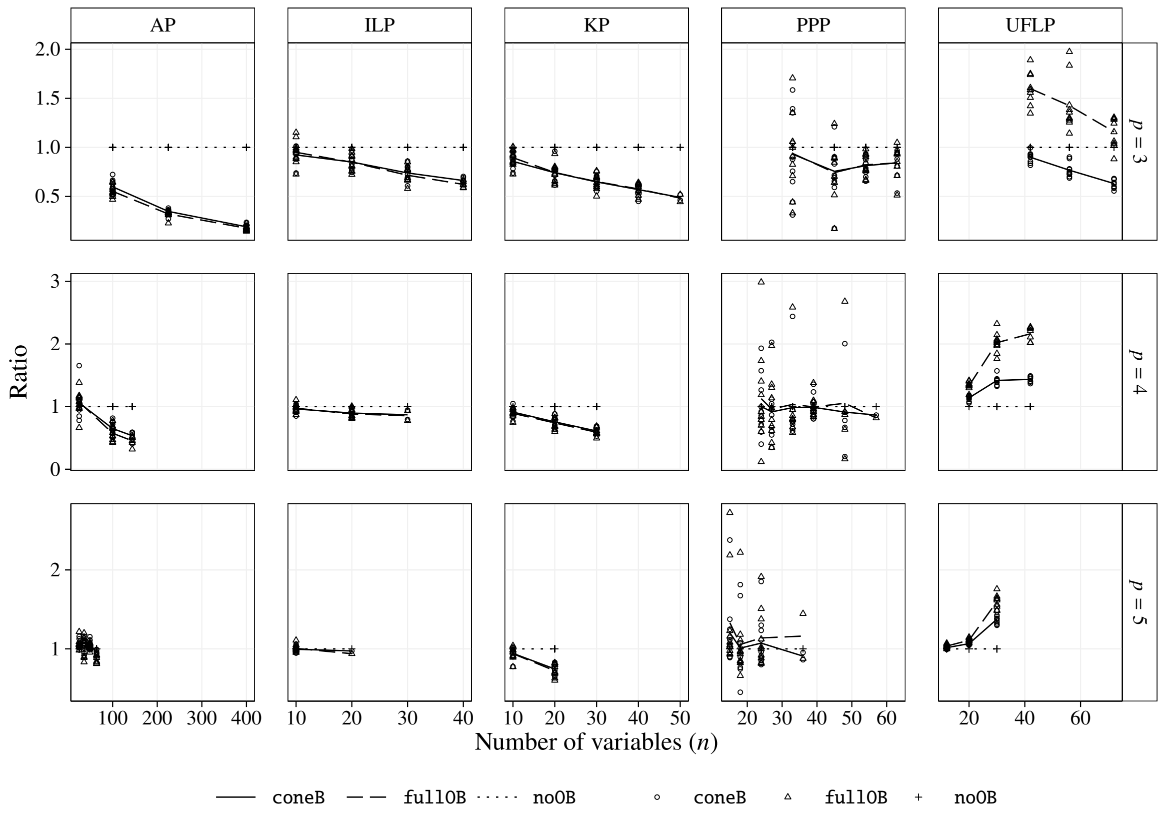


Figure 6: CPU time ratio (CPU time divided with the CPU using noOB) for each test instance (points) together with averages (lines).

5.2 Performance of the different algorithm configurations

A comparison of the different algorithm configurations is given in Figure 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.

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.

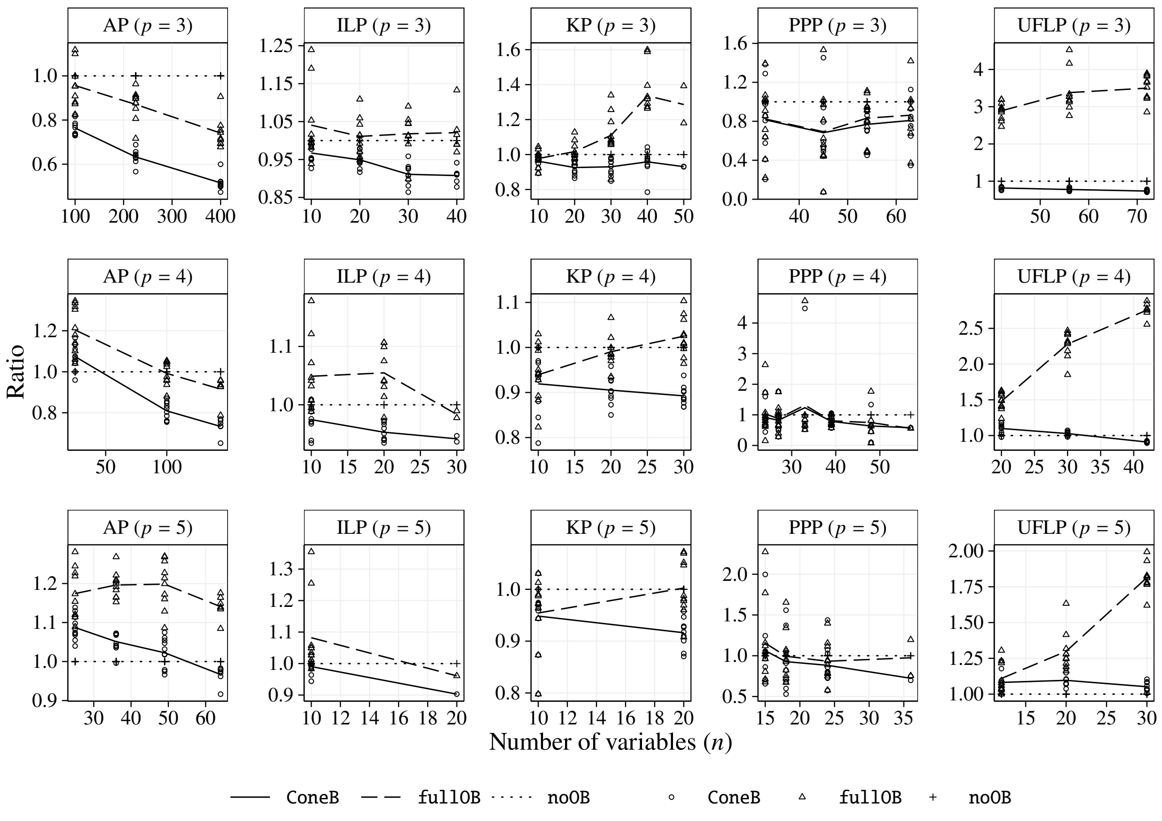


Figure 7: Nodes in the branching tree ratio (number of nodes divided with the number of nodes using noOB) for each test instance (points) together with averages (lines).

The node ratio of the branching tree is depicted in Figure 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 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. (2021)).

- The way nodes are fathomed in the tree is significantly impacted by objective branching (either coneB or fullOB). Figure 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

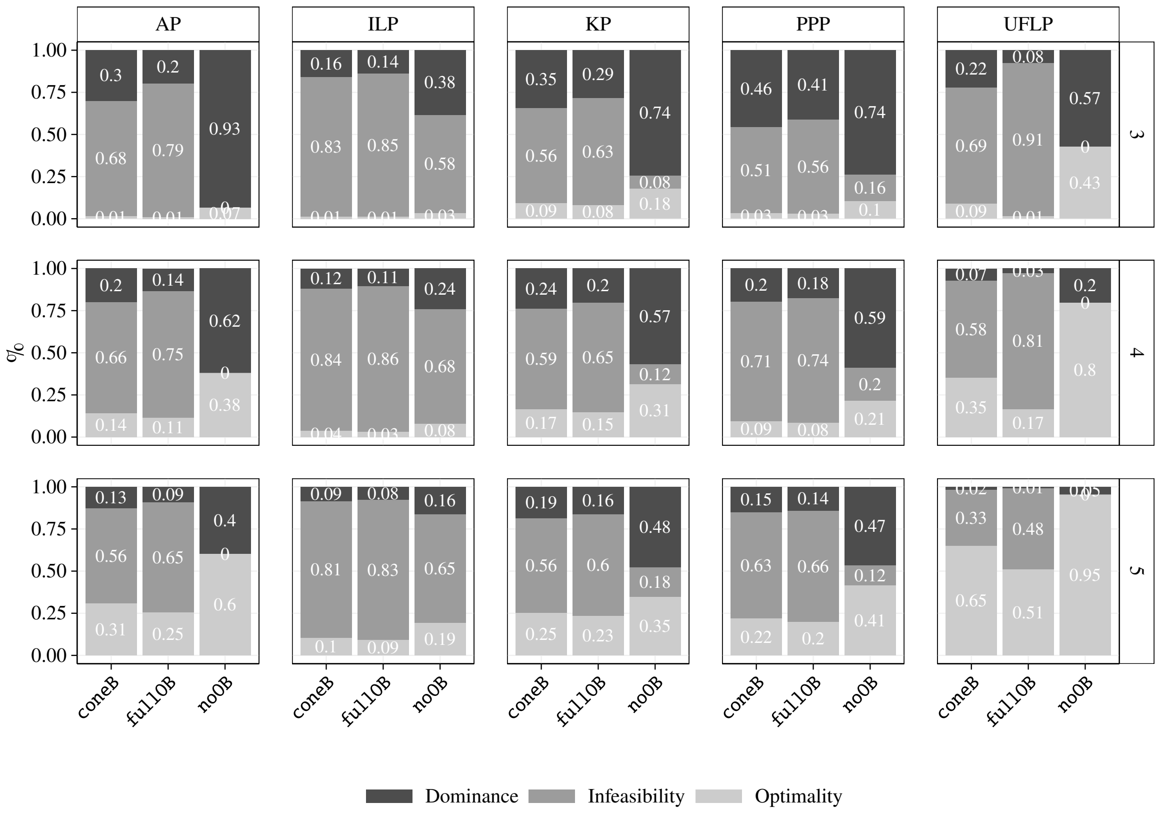


Figure 8: Proportion of leaf nodes fathomed by dominance, infeasibility, and optimality.

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, while it can be stopped when a dominated one is found with `noOB`.
- Due to Theorem 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,

Table 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.

	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.

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. Moreover, these negative effects on CPU increase with increasing number of objectives as can be seen in Figure 6 for PPP and UFLP which indeed are instances with a high number of non-dominated points.

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 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 in 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 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

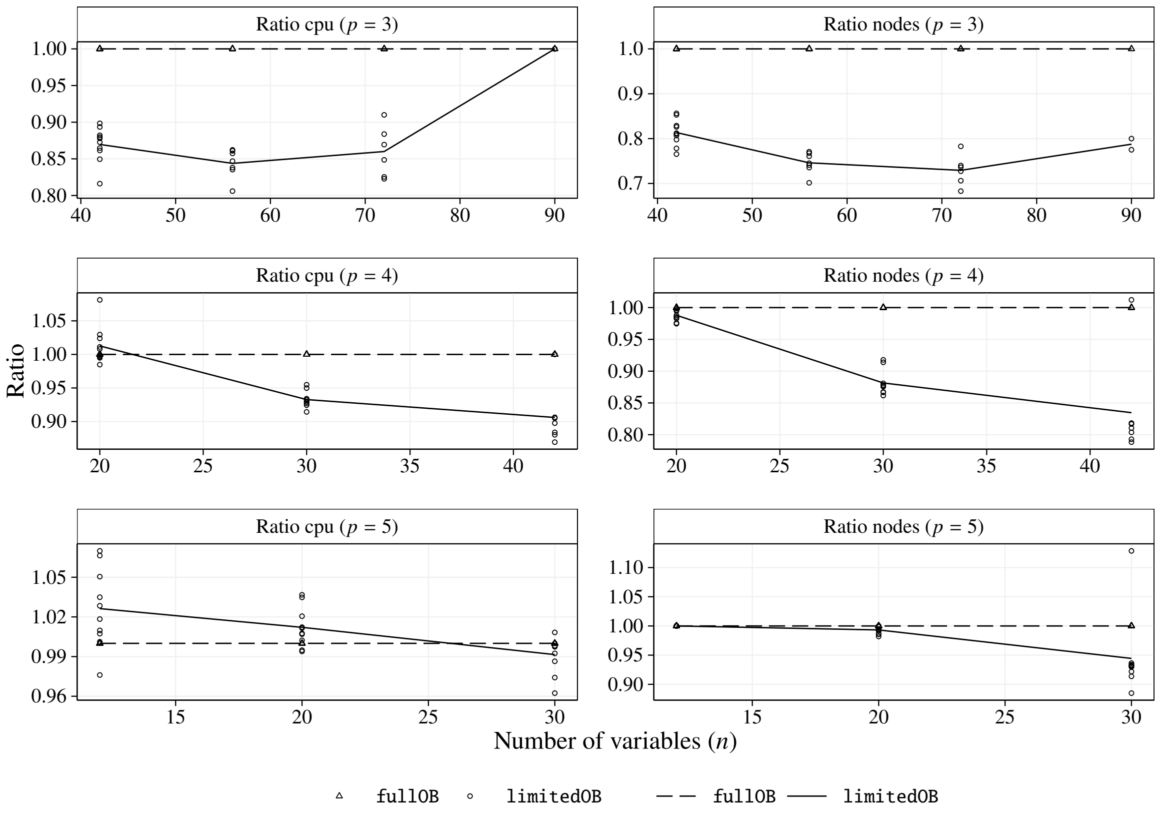


Figure 9: Ratios of comparing CPU and nodes in the tree using `fullOB` as benchmark (in the denominator when dividing the numbers) for UFLP instances.

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 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 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 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 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.

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, for example, 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 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 features like, for example, providing starting solutions to CPLEX and re-ordering the subproblems in a smart way. This clearly leads to further improvements in running time, however, the general concept is very similar to that of OSS-DFK. Holzmann and Smith (2018) use weighted Tchebychev scalarizations rather than e-constraint scalarizations and present promising results, particularly when using an analogous subproblem structure as Klamroth et al. (2015), i.e., the same structure as in OSS-DFK. 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 10. In general, the OSS algorithms 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-

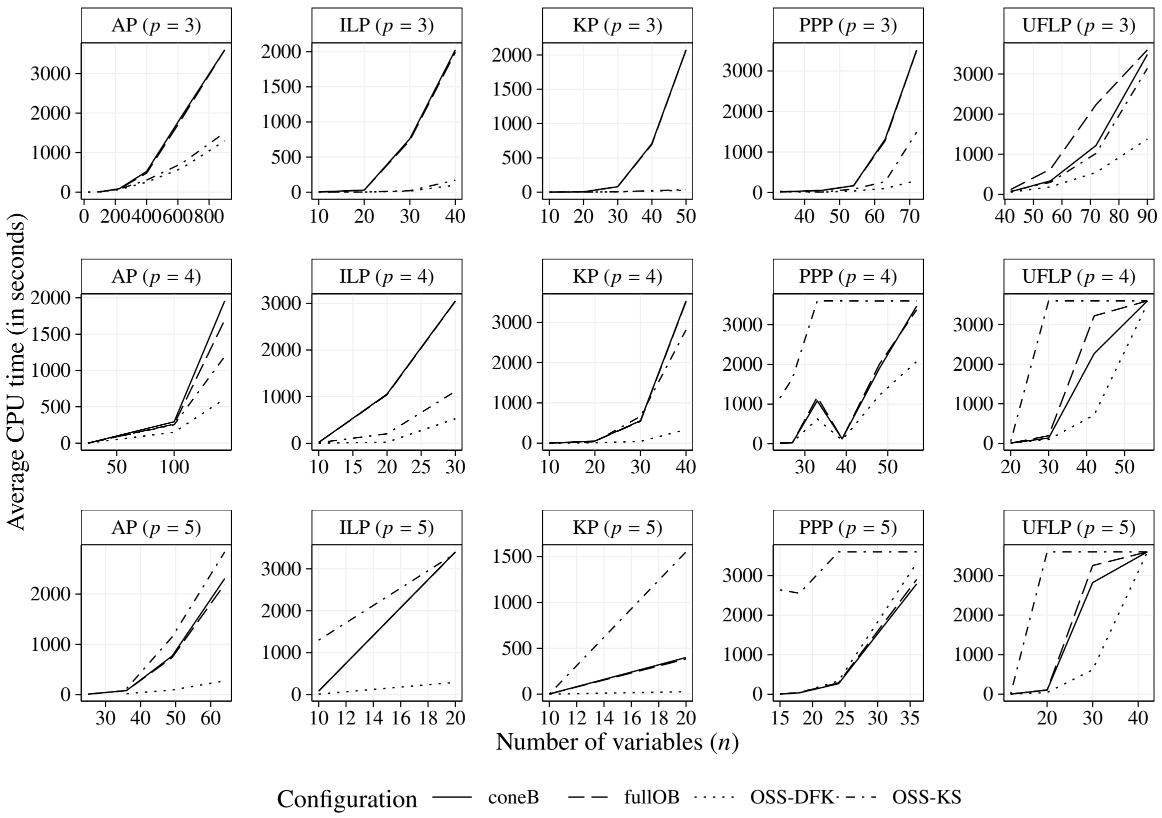


Figure 10: Average cpu times, expressed in seconds, in function of the number of variables for each number of objectives and problem classes. Unsolved instances are included here. Four configurations are depicted: `fullOB`, `coneB`, `OSS-DFK`, and `OSS-KS`.

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.

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 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.

References

Nathan Adelgren and Akshay Gupte. Branch-and-bound for biobjective mixed-integer linear programming. *INFORMS Journal on Computing*, oct 2021. doi:https://doi.org/10.1287/ijoc.2021.1092 . Published online in Articles in Advance 21 Oct 2021.

Tolga Bektaş. Disjunctive programming for multiobjective discrete optimisation. *INFORMS Journal on Computing*, 30(4):625–633, nov 2018. doi:https://doi.org/10.1287/ijoc.2017.0804 .

Pietro Belotti, Banu Soylu, and Margaret M. Wiecek. A branch-and-bound algorithm for biojbective mixed-intger programs. Technical report, Clemson University, 2013.

H. P. Benson. 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, 1998.

N. Boland and H. Charkhgardand M. Savelsbergh. The l-shape search method for triobjective integer programming. *Mathematical Programming Computation*, 8(2):217–251, Jun 2016. doi:10.1007/s12532-015-0093-3 .

N. Boland, H. Charkhgard, and M. Savelsbergh. The quadrant shrinking method: A simple and efficient algorithm for solving tri-objective integer programs. *European Journal of Operational Research*, 260(3):873 – 885, 2017. ISSN 0377-2217. doi:10.1016/j.ejor.2016.03.035 .

L. Csirmaz. Using multiobjective optimization to map the entropy region. *Computational Optimization and Applications*, 63(1):45–67, jun 2015. doi:10.1007/s10589-015-9760-6 .

K. Dächert and K. Klamroth. A linear bound on the number of scalarizations needed to solve discrete tricriteria optimization problems. *Journal of Global Optimization*, 61(4):643–676, Apr 2015. doi:10.1007/s10898-014-0205-z .

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

M. Ehrgott and X. Gandibleux. Bound sets for biobjective combinatorial optimization problems. *Computers & Operations Research*, 34(9):2674–2694, 2007. doi:10.1016/j.cor.2005.10.003 .

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

N. Forget. C++ implementation of multi-objective branch-and-bound. `https://github.com/NicolasJForget/LinearRelaxationBasedMultiObjectiveBranchAndBound/releases/tag/v2.0`, 2021.

N. Forget, L. R. Nielsen, and S. L. Gadegaard. Examples of lower and upper bound sets (MOrepo-Forget20), 2020a. URL `https://mcdmsociety.github.io/MOrepo-Forget20/report_interactive.html`.

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

N. Forget, L. R. Nielsen, and S. L. Gadegaard. Instances and results for linear relaxation based branch-and-bound (MOrepo-Kirlik14), 2020c. URL `https://github.com/MCDMSociety/MOrepo-Forget21`.

N. Forget, S.L. Gadegaard, and L.R. Nielsen. Linear relaxation based branch-and-bound for multi-objective integer programming with warm-starting. `http://www.optimization-online.org/DB_FILE/2021/08/8531.pdf`, aug 2021. Preprint.

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

A. H. Hamel, A. Löhne, and B. Rudloff. Benson type algorithms for linear vector optimization and applications. *Journal of Global Optimization*, 59(4):811–836, aug 2013. doi: 10.1007/s10898-013-0098-2 .

T. Holzmann and J.C. Smith. Solving discrete multi-objective optimization problems using modified augmented weighted Tchebychev scalarizations. *European Journal of Operational Research*, 271: 436–449, 2018. doi:10.1016/j.ejor.2018.05.036 .

G. Kirlik. Test instances for multiobjective discrete optimization problems, 2014. URL `http://home.ku.edu.tr/~moolibrary/`.

G. Kirlik and S. Sayın. A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems. *European Journal of Operational Research*, 232(3):479 – 488, 2014. ISSN 0377-2217. doi:10.1016/j.ejor.2013.08.001 .

G. Kiziltan and E. Yucaoğlu. An algorithm for multiobjective zero-one linear programming. *Management Science*, 29(12):1444–1453, December 1983. doi:10.1287/mnsc.29.12.1444 .

K. Klamroth, R. Lacour, and D Vanderpooten. On the representation of the search region in multi-objective optimization. *European Journal of Operational Research*, 245:767–778, 2015. doi: 10.1016/j.ejor.2015.03.031 .

D. Klein and E. Hannan. An algorithm for the multiple objective integer linear programming problem. *European Journal of Operational Research*, 9(4):378 – 385, 1982. doi:10.1016/0377-2217(82) 90182-5 .

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

G. Mavrotas and D. Diakoulaki. A branch and bound algorithm for mixed zero-one multiple objective linear programming. *European Journal of Operational Research*, 107(3):530–541, 1998. doi: 10.1016/S0377-2217(97)00077-5 .

G. Mavrotas and D. Diakoulaki. 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, 2005. doi:10.1016/j.amc.2005.01.038 .

M. Ozlen, B.A. Burton, and C.A.G. MacRae. Multi-objective integer programming: An improved recursive algorithm. *Journal of Optimization Theory and Applications*, 160(2):470–482, Feb 2014. doi:10.1007/s10957-013-0364-y .

S.N. Parragh and F. Tricoire. Branch-and-bound for bi-objective integer programming. *INFORMS Journal on Computing*, 31(4):805–822, 2019. doi:10.1287/ijoc.2018.0856 .

A. Przybylski and X. Gandibleux. Multi-objective branch and bound. *European Journal of Operational Research*, 260(3):856 – 872, 2017. doi:10.1016/j.ejor.2017.01.032 .

R. M. Ramos, S. Alonso, J. Sicilia, and C. González. The problem of the optimal biobjective spanning tree. *European Journal of Operational Research*, 111(3):617 – 628, 1998. doi:10.1016/ S0377-2217(97)00391-3 .

Marianna De Santis, Gabriele Eichfelder, Julia Niebling, and Stefan Rocktäschel. Solving multiobjective mixed integer convex optimization problems. *SIAM Journal on Optimization*, 30(4):3122–3145, jan 2020. doi:https://doi.org/10.1137/19M1264709 .

F. Sourd and O. Spanjaard. A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem. *INFORMS Journal on Computing*, 20(3):472–484, 2008. doi: 10.1287/ijoc.1070.0260 .

T. Stidsen and K. A. Andersen. A hybrid approach for biobjective optimization. *Discrete Optimization*, 28:89–114, 2018. doi:10.1016/j.disopt.2018.02.001 .

T. Stidsen, K. A. Andersen, and B. Dammann. A branch and bound algorithm for a class of biobjective mixed integer programs. *Management Science*, 60(4):1009–1032, 2014. doi:10.1287/mnsc.2013.1802 .

J. Sylva and A. Crema. 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, 2004. doi: 10.1016/S0377-2217(03)00255-8 .

S. Tamby and D. Vanderpooten. Enumeration of the nondominated set of multiobjective discrete optimization problems. *INFORMS Journal on Computing*, 33(1):72–85, 2021. doi:10.1287/ijoc.2020.0953 .

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

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

T Vincent. *Caractérisation des solutions efficaces et algorithmes d'énumération exacts pour l'optimisation multiobjectif en variables mixtes binaires*. PhD thesis, LINA, Université de Nantes, France, 2013. URL `http://www.theses.fr/2013NANT2065`.

T. Vincent, F. Seipp, S. Ruzika, A. Przybylski, and X. Gandibleux. Multiple objective branch and bound for mixed 0-1 linear programming: Corrections and improvements for the biobjective case. *Computers & Operations Research*, 40(1):498–509, 2013. doi:10.1016/j.cor.2012.08.003 .

M. Visée, J. Teghem, M. Pirlot, and E. L. Ulungu. Two-phases method and branch and bound procedures to solve the bi–objective knapsack problem. *Journal of Global Optimization*, 12(2): 139–155, 1998. doi:10.1023/A:1008258310679 .