Branch-and-bound and objective branching with three 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 Sciences and Technologies, University of Nantes, 2 Rue de la Houssinière BP 92208, 44322 Nantes Cedex 03 - France.

December 15, 2020

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. Besides the classical dominance test, bound sets are used 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 tri-objective combinatorial optimization problems. Several difficulties arise in this case, as there is no longer a lexicographic order among nondominated outcome vectors in the multi-objective case, with more than two 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 subproblems. Numerical experiments on tri-objective knapsack, assignment and facility location problems show a significant speed-up in the B&B framework.

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

1 Introduction

In many real-world problems, it is difficult to define only one objective to optimize. Indeed, a decision maker may be interested in minimizing different objectives, e.g. operational costs, as well as 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 must 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 combinatorial optimization* (MOCO) problems, where all variables are binary. In this paper we will consider multi-objective optimization problems with three objective functions that must be optimized simultaneously. Although most research has

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

focused on the bi-objective MOCO (only two objective functions), the interest in MOCOs in higher dimensions has risen during the last 10-20 years. We review the state-of-the-art literature on MOCO problems in the following.

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 MOCO problem and then solving a series of single objective optimization problems (SOP), thus utilizing the power of modern commercial integer programming (IP) solvers such as CPLEX or Gurobi.

A rather straightforward approach for MOCO problems is 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, Dächert and Klamroth (2015) and Klamroth, Lacour, and Vanderpooten (2015) for a decomposition into pairwise non-redundant search zones respectively in three and in arbitrary dimensions, Tamby and Vanderpooten (2020) for an efficient strategy to enumerate these search zones using ε-constraint scalarizations, Kirlik and Sayın (2014) and Boland, Charkhgard, and Savelsbergh (2017) for methods based on ε-constraint scalarizations in combination with dimension reduction respectively for the multi and tri-objective case, Boland and Savelsbergh (2016) for the L-shaped search method for problems with three objectives and references therein.

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 a 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 is developed by Klein and Hannan (1982). The algorithm uses post optimality techniques to solve a series of integer programs all in one tree structure. In the 1980s and 1990s only a few papers on multi-objective B&B algorithms are 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 GonzÃ¡lez (1998); Ulungu and Teghem (1997, 1995), and Visée, Teghem, Pirlot, and Ulungu (1998).

During the last 20 years more attention has been given to decision space search methods: Mavrotas and Diakoulaki (1998) develop 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 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 (2019) 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) that test additional lower bound sets, such as the linear and convex relaxations. 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 term *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 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, while 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).

Despite the progress in decision space search algorithms designed for general bi-objective combinatorial and mixed integer programming, there has been no works tackling the multi-objective case using decision space search algorithms. One reason for this is that objective space branching, such as Pareto branching, is not easily extended to higher dimensions. A second reason is that comparing upper and lower bound sets in higher dimensions is not trivial either. In this paper we propose a novel B&B algorithm for solving multi-objective combinatorial optimization problems using a "one-tree" search to generate all non-dominated outcomes. The algorithm uses the non-dominated outcomes of the multi-objective LP relaxation as a lower bound set of each branching node, and then this lower bound set is compared to the set of currently non-dominated feasible objective vectors found so far in the search. The main contributions of the paper are fourfold:

1. We propose the first general B&B framework for multi-objective combinatorial optimization problems.

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 tri-objective combinatorial optimization problems showing different 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* $\mathcal{X} = \{x \in \{0,1\}^n \mid Ax \geqq b\}$ 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 $\mathcal{X}^{LP} = \{x \in [0,1]^n \mid Ax \geqq b\}$.

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 set of points $\mathcal{S} \subseteq \mathbb{R}^p$, it is possible to define lower and upper bound sets for $\mathcal{S}_N$. For this purpose, 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 it 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 $\mathcal{U} = \mathcal{U}_N$.*

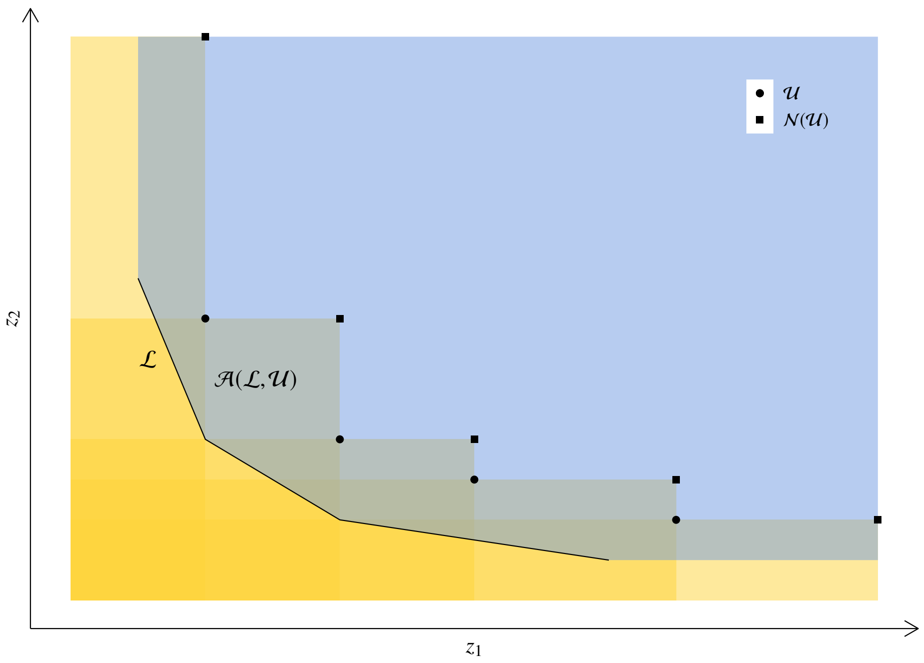


Figure 1: Given a lower bound set $\mathcal{L}$ (the line segments) and an upper bound set $\mathcal{U}$, the search region $\mathcal{A}(\mathcal{L},\mathcal{U})$ is given by the closure of the gray area.

Two specific lower and upper bound sets respectively of $\mathcal{Y}_N$ are the ideal point y^I satisfying

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

and the nadir point y^N satisfying

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

Given an upper bound set $\mathcal{U}$ and a lower bound set $\mathcal{L}$, we say that $\mathcal{L}$ is *fully weakly dominated* by $\mathcal{U}$ if $\forall l \in \mathcal{L}$, $\exists u \in \mathcal{U}$ such that $u \leqq l$. Furthermore, we say that $\mathcal{L}$ is *partially dominated* by $\mathcal{U}$ if $\mathcal{L}$ is not fully weakly dominated by $\mathcal{U}$ and there exists at least one pair of points (l,u) such that $l \in \mathcal{L}$, $u \in \mathcal{U}$ and $u \leqslant l$.

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 of 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 in Klamroth et al. (2015) will be used and be extended 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 grey and yellow areas in Figure 1). This region corresponds to the part of the objective space that is not dominated by

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

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 $\mathcal{U}$, $\mathcal{N}(\mathcal{U})$, is a unique discrete set of points in $\mathbb{R}^p$ satisfying*

1. $\mathcal{A}(\mathcal{U}) = \bigcup_{u \in \mathcal{N}(\mathcal{U})} C(u)$ *and*

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

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 $\mathcal{Y}_N \subset \mathcal{L} + \mathbb{R}^p_{\geqq}$ (gray 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 gray areas in Figure 1). In the remainder of the paper we assume that $\mathcal{U} \subseteq \mathcal{Y}$, thus the boundary of $\mathcal{A}(\mathcal{U})$ is dominated, and hence we define the *search region* as $\mathcal{A}(\mathcal{L}, \mathcal{U}) = (\mathcal{L} + \mathbb{R}^p_{\geqq}) \cap (\mathcal{N}(\mathcal{U}) - \mathbb{R}^p_{\geqq})$.

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 set of efficient solutions. Furthermore, only binary variables are considered, as many real world problems can be described using only binary variables. However, all the concepts exposed in this paper easily extend to problems with integer variables.

The principle 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 where 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 it contains 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, containing the initial problem P. The upper bound set (on the set of non-dominated points of the initial problem P) $\mathcal{U}$ is initialized as an empty set. It may be initialized with feasible solutions if some are known prior to the resolution, but it will not be the case in this paper. The outline of the multi-objective B&B is given in Algorithm 1.

The purpose of Step 1 is to select a node η that will be explored. For this purpose, depth-first strategies are widely used in the literature for multi-objective B&B (see Przybylski and Gandibleux (2017)), and it is shown in Visée et al. (1998) that on their set of test instances, a depth-first strategy performs better than a breadth-first strategy. However, as some key components of this framework are different from theirs, a breadth first strategy will also be tested. We refer the reader to Section 5 for further experiments.

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, it corresponds to the non-dominated part of a convex polytope of dimension at most p. In order to compute the linear relaxation, Benson's outer approximation algorithm (Benson, 1998) will be used. It has two main advantages. First, it computes $\mathcal{Y}_N^{LP}(\eta) := (C\mathcal{X}^{LP}(\eta))_N$ instead of $\mathcal{X}_E^{LP}(\eta)$ as is the case, for example, with the multi-criteria simplex method (Ehrgott, 2005), which may be significantly cheaper in terms of computation time as different efficient solutions may have the same objective vector. Then, the output of this algorithm both gives a representation of $\mathcal{L}(\eta)$ in terms of its extreme points, which will be of interest for Step 3 and Step 5; and a representation of $\mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}$ in terms of hyperplanes (defining the facets of the polytope $\mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}$), which will be of interest for Step 3 and Step 4.

Three cases may arise in Step 3. 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, 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 apply this test, Proposition 1, which is a generalization of Proposition 1 from Gadegaard et al. (2019), will be used. 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, as by definition a local upper bound $u \in \mathcal{N}(\mathcal{U})$ 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 $(\mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}) \cap \mathcal{N}(\mathcal{U}) = \emptyset$.*

Proof. Assume that $(\mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}) \cap \mathcal{N}(\mathcal{U}) = \emptyset$ is true, i.e., there is no $u \in \mathcal{N}(\mathcal{U})$ such that $u \in \mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}$. Suppose that $\mathcal{A}(\mathcal{L}(\eta), \mathcal{U}) \neq \emptyset$, i.e., there exists at least one $y \in \mathcal{A}(\mathcal{L}(\eta), \mathcal{U})$. In particular, by definition of the search region, we have $y \in (\mathcal{N}(\mathcal{U}) - \mathbb{R}^p_{\geqq})$, which implies that there exists $u \in \mathcal{N}(\mathcal{U})$ such that $u \in \{y\} + \mathbb{R}^p_{\geqq}$. Furthermore, by definition of the search region, we also have that $y \in \mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}$ and thus, $u \in \mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}$, which contradicts the initial statement. □

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 $y \in \mathbb{R}^p$ is located in $\mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}$ if for each $i \in \{1, ..., F\}$, $n^{iT} \cdot y \geq d^i$ 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 $n^{iT} \cdot u < d^i$.*

Proof. By Proposition 1, the node η can be fathomed if $(\mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}) \cap \mathcal{N}(\mathcal{U}) = \emptyset$, i.e., if there is no $u \in \mathcal{N}(\mathcal{U})$ such that $u \in \mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}$. A point $y \in \mathbb{R}^p$ is not in $\mathcal{L}(\eta) + \mathbb{R}^p_{\geqq}$ if there is at least one $i \in \{1, ..., F\}$ such that $n^{iT} \cdot y < d^i$. Hence, if for each $u \in \mathcal{N}(\mathcal{U})$, there exists at least one $i \in \{1, ..., F\}$ such that $n^{iT} \cdot u < d^i$, the node η can be fathomed by dominance. $\square$

If none of the three cases described above arise, 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 exist. 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 exists 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 the decision space (Step 5), the subproblems are created by selecting a free variable x_i, $i \in \{1, ..., n\}$, and by fixing it to either 0 or 1. Hence, at a node η, the problem $P(\eta)$ will be split into subproblems $P(\eta_k)$ such that $\mathcal{X}(\eta_k) = \{x \in \mathcal{X}(\eta) \mid x_i = k\}$, where $k \in \{0, 1\}$. Consequently, a variable x_i is said to be free at node η if it is not fixed by a constraint in the form $x_i = 0$ or $x_i = 1$. The choice of the free variable to branch on at a given node is discussed in Section 5.

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, such as for example slicing (see Stidsen et al. (2014), Stidsen and Andersen (2018)). In this study, the focus will be on objective branching as defined in Stidsen et al. (2014), Gadegaard et al. (2019), and Parragh and Tricoire (2019) (alternatively called Pareto branching or Extended Pareto branching in some of these references). The reason for this choice is that the way it is defined naturally extends to an arbitrary number of objectives p. This definition is given in Definition 3, and in the remainder of this paper, the notion of objective branching will always refer to this definition.

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.*

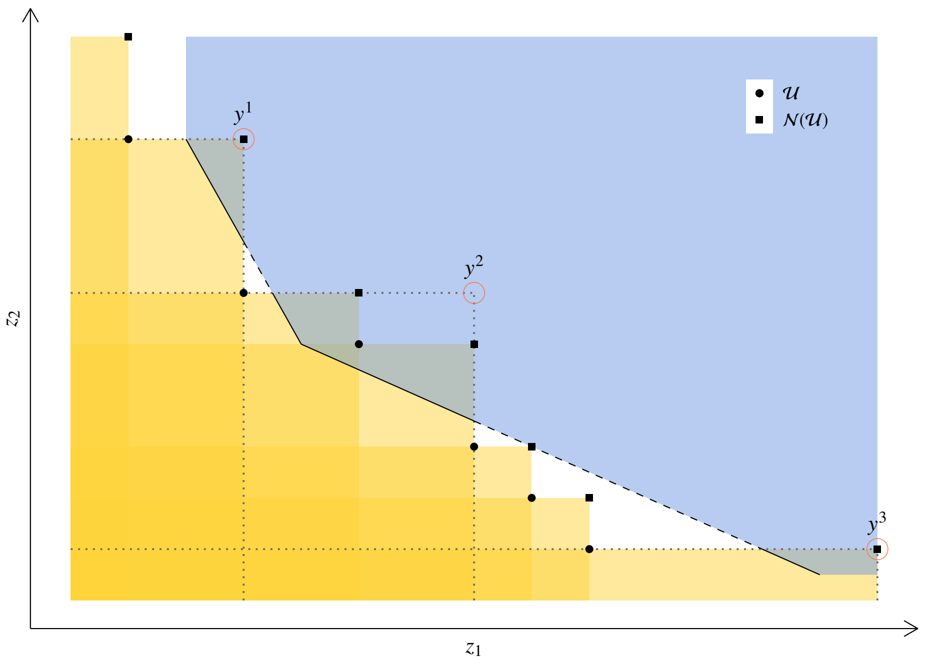


Figure 2: A lower bound set (solid and dashed lines) partially dominated by the upper bound set. The dominated area of the lower bound set is represented by the dashed lines. Three disjoint subproblems are created in the objective space by applying objective branching on y^1, y^2 and y^3, represented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

Now consider the situation depicted in Figure 2, where the node η 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...) exploring 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 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. Two important properties (1.(a) and 1.(b)) are the main focus in the paper, but a third property (1.(c)) will be briefly discussed in the rest of this section and in Section 5. Note that these properties are ordered, i.e., Property 1.(a) is more important than Property 1.(b), which again is more important than Property 1.(c). Note that in particular, Property 1.(b) and 1.(c) are conflicting, which is why such an order is required. 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 in problem $P(\eta, s)$) and ρ be the number of subproblems created in the objective space at node η.

Property 1. *The desirable properties for objective branching are the following:*

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

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

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

Property 1.(a) 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 thus 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 1.(b) 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. In this sense, objective branching should be sparse.

Property 1.(c) 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. In this sense, objective branching should be tight.

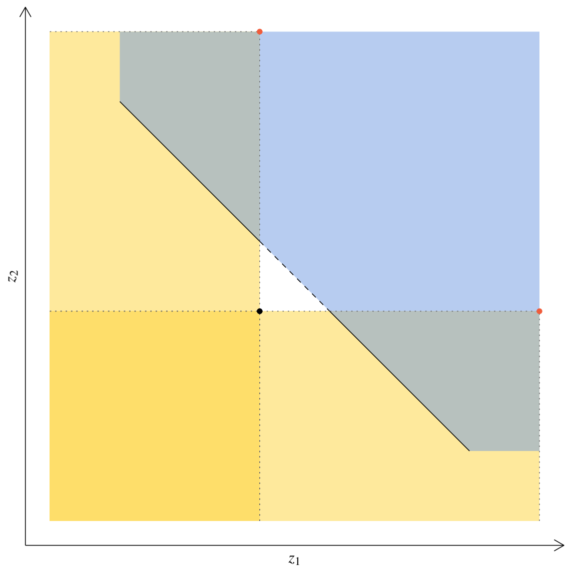
Note that in this definition of objective branching, no constraint in the form $z_k(x) \geqq \bar{z}_k$ is used because, as explained in Stidsen et al. (2014), it often leads to many degenerate pivots when solving the linear programming relaxation, and thus it increases the computation time significantly.

4.1 The complications of going from two to three objectives

Several approaches have been developed in the literature for identifying the subproblems to be created. The first one was initially proposed by Stidsen et al. (2014) and 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 is made 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, defined by a unique point, is represented by the black point. 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 (given by the three red points) as in the bi-objective case, the subproblems created will have redundancies between the subproblems (each subproblem defines a gray search area). Every point in the red areas is included in the search area of more than one subproblem and thus, it will not satisfy Property 1.(b).

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). The authors 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.



(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 search areas (gray) satisfy Property 1.

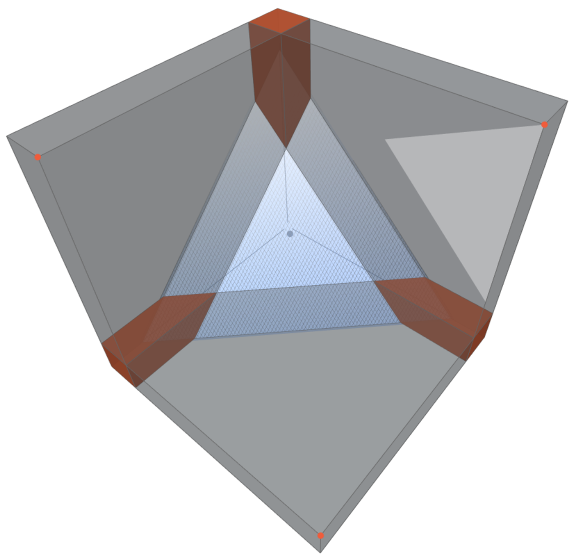

(b) Three-objective case: The dominated part of the lower bound set is the blue area in the middle. When objective branching is applied on each local upper bound, there are redundancies between the subproblems (each subproblem defines a gray search area). Every point in the red areas is included in the 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. (2020c).

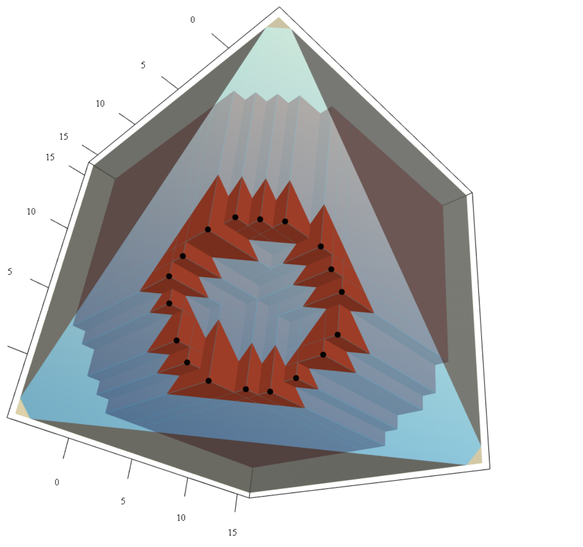
However, having the lower bound set split is not a sufficient condition to apply objective branching with the desirable properties in the three-objective case. For example, in Figure 4, the lower bound set consists of a unique facet represented by the gray surface. It is partially dominated by the upper bound set (the dominated area is depicted in black). It can be observed that there are two non-dominated connected components here. However, if objective branching is applied on each component like in the bi-objective case (in Figure 4b, 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), one of the subproblems obtained is fully included in the other subproblem and thus, there will be redundancies. Hence, Property 1.(b) is not satisfied.

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 A strategy for computing 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. This section also considers Property 1.(c).

The strategy is based on merging operation of redundant subproblems, which are defined by the


(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.

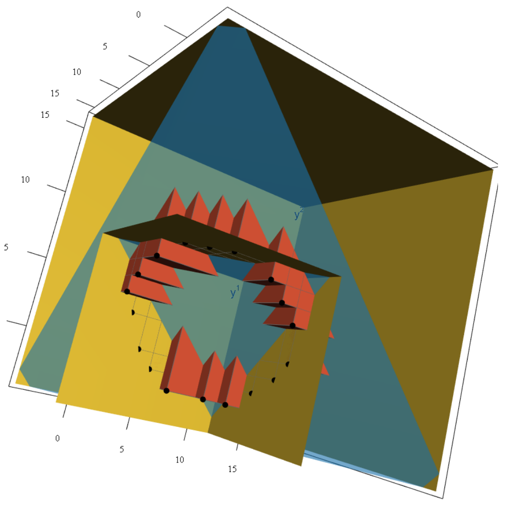

(b) Objective branching applied to y^1 and y^2. One subproblem is fully included in the other subproblem.

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

local upper bound 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 or not at each node.

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 such that $s^I_k = \min(s^1_k, s^2_k)$, $\forall k \in \{1, ..., p\}$. The cone $C(s^I)$ will be called* the intersection cone.

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 subproblems 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 $u^1, ..., u^h \in \mathcal{D}(\eta)$. The point $s \in \mathbb{R}^p$ is a super local upper bound of the local upper bounds $u^1, ..., u^h$ if $s_k = \max_{i \in \{1,...,h\}} u^i_k$, $\forall k \in \{1, ..., p\}$. Furthermore, if a local upper bound $u \in \mathcal{D}(\eta)$ satisfies $u \subset C(s)$, it is said that u is contained in s.*

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

To conclude, given two local upper bounds $u^1, u^2 \in \mathcal{D}(\eta)$, the goal is to know whether it is possible to apply objective branching on u^1 and u^2 with the desirable properties, or if only one large subproblem should be considered by applying objective branching on a super local upper bound s that contains u^1 and u^2. As explained previously, u^1 and u^2 will be merged if their intersection point u^I is dominated by the current lower bound set $\mathcal{L}(\eta)$. An overview of a merging operation is given in Algorithm 2. 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.

Algorithm 2 Merge(u^1, u^2)

```
1: Input :
2: $u^1, u^2$ : two (super) local upper bounds
3: Algorithm :
4: for $k = 1, ..., p$ do
5:     $s_k = \max(u^1_k, u^2_k)$
6: end for
7: return $s$
8: Output :
9: $s$ : 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:*

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

2.(b) 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* $\mathcal{L}(\eta)$.

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

2.(d) The size of $\mathcal{S}$ *is maximal, i.e., we have the maximum number of super local upper bounds in S that satisfy the properties 1.(a), 1.(b), and 1.(c).*

Algorithm 3 Computation of the set of super local upper bounds

```
1:  Input :
2:  $\mathcal{D}(\eta)$ : set of dominated local upper bounds at node $\eta$
3:  a lower bound set $\mathcal{L}(\eta)$
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
```

Property 2.(a) implies that each local upper bound is merged in at least one super local upper bound or is a super local upper bound itself. For the objective branching, this implies that no solution is overlooked. Indeed, all the areas where non-dominated points can be found are included in a super local upper bound and thus, Property 1.(a) is satisfied.

Property 2.(b) 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 1.(b) is satisfied.

Property 2.(c) 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 means that in the subproblems created with objective branching, as much of the dominated region as possible is discarded. Hence, this ensures that Property 1.(c) is satisfied.

Property 2.(d) 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. Otherwise, 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 1.(c).

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, this algorithm needs the set of dominated local upper bounds $\mathcal{D}(\eta)$ and the corresponding lower bound set $\mathcal{L}(\eta)$ as input. The algorithm is described in Algorithm 3. The function $Merge(s^1, s^2)$ simply merges s^1 and s^2 as presented in Section 4.2.1.

Theorem 1. *Algorithm 3 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 2.(a) is satisfied.

Algorithm 3 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 2.(b) 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 2.(c) is satisfied.

In order to violate the Property 2.(d), 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 3. 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 3 runs in* $O(|\mathcal{D}(\eta)|^3 \cdot f(\mathcal{L}(\eta)))$.

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

Each time the algorithm enters its main loop, it needs to identify a pair to merge. For this purpose, a simple pairwise comparison of each element of $\mathcal{S}_t$ can be done, where $\mathcal{S}_t$ is the set $\mathcal{S}$ at iteration t of Algorithm 3. 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 comparison (this bound is not tight) and each pairwise comparison involves a dominance test. Algorithm 3 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 2.(b). 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 $L(\eta)$.*

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 Properties 2.(a), 2.(b) and 2.(c), and therefore Property 2.(d) 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 2.(c) the super local upper bounds are as tight as possible, and thus $s^1 \leqq s^2$. 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 2.(b) 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 $|\mathcal{S}| \geq 2$, since $\mathcal{S}$ satisfies Property 2, and in particular Property 2.(a), 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 an $u' \in \mathcal{D}(\eta, s^1)$ 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 an $u'' \in \mathcal{D}(\eta, s^2)$ 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 2.(b). If this is not the case, the same principle applies to u' and u''. As Lemma 2, Lemma 3 and $s^1 \neq s^2$ holds true, the situation where s^1 and s^2 have to be merged will always be reached, and this will contradict Property 2.(b).

□

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 situation 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 $u \in \mathcal{D}(\eta, s')$ such that $u \notin \mathcal{D}(\eta, s)$ 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 u), which contradicts the fact that $\mathcal{S}$ satisfies Property 2.(b). 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 exist $\mathcal{D}(\eta, \hat{s}')$ 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 2.(b). The same reasoning can be applied to v' and $\mathcal{S}$. □

4.3 An alternative strategy : objective branching in a single cone

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 1.(c). In this section, this property will be relaxed.

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 many nodes are generated and the algorithm fails to explore them fast enough. In this case, a new question arises: is it possible to derive inequalities from the partial dominance of the lower bound set without generating more subproblems?

Doing so is equivalent to always creating a unique subproblem in the objective space, and compared to a B&B algorithm without objective branching, we actually do not generate additional subproblems as only two are created due to the variable splitting in the decision space.

Algorithm 3 will be reused, except that a unique super local upper bound should be obtained at each node η and therefore, all super local upper bounds will always be merged. Due to the way the super local upper bounds are merged, the final super local upper bound is actually the nadir point of $\mathcal{D}(\eta)$. Let $d^N(\eta)$ denote this point. In this alternative strategy, Step 4 of Algorithm 1 becomes *apply objective branching on $d^N(\eta)$*.

5 Computational experiments

In this section we report on the computational experiments conducted with the tri-objective Branch-and-Bound (B&B) algorithm.

All instances are converted to minimization problems meaning that if an objective function $z(x)$ should be maximized, $-z(x)$ is minimized instead.

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

1. How do the different algorithm configurations perform, and which configurations perform the best? In particular, is it worth to apply objective branching?
2. Why does objective branching perform the way it does?
3. Which node selection strategy performs the best?
4. How is the performance of the tri-objective B&B algorithm compared to an objective space search algorithm?

5.1 Implementation details and algorithm configurations

All algorithms were implemented in Julia 1.0.1. The experiments were carried out on a computer with an Intel(R) Core(TM) i7-4785T CPU @ 2.20GHz processor and 16GB of RAM memory, using Linux Ubuntu 14.04 LTS.

In order to compute the linear relaxation at each node, the solver Bensolve was used (Löhne and Weißing, 2020). To avoid reading and writing text files at each node of the tree, we have implemented a wrapper that calls Bensolve, retrieves some outputs, and inserts them directly in our code as matrices. Numerical instabilities occurred during the computation of lower bound sets, leading to missing non-dominated points in the final output. The matrix used when finding the normal to each hyperplane of a lower bound set may in a few cases be close to singular (its determinant is close to zero). We therefore introduced a small value $\epsilon > 0$ so that if the determinant is less than or equal to ϵ, then the hyperplane is discarded, thus leaving a weaker but valid lower bound set. For more information, we refer the reader to the section about numerical instabilities in Forget, Nielsen, and Gadegaard (2020d). We used $\epsilon = 0.001$ in all tests reported in this paper. With this parameter setting, no non-dominated points were missed.

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 is the most different (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 case 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 variable with the smallest index is chosen.

To test different algorithm configurations we use the following parameters:

- Two node selection methods are considered (Step 1 of Algorithm 1):

  `B`: breadth-first strategy.

  `D`: depth-first strategy.

- Three objective branching strategies are considered:

  `N`: no objective branching is performed. This is the equivalent of skipping Step 4 of Algorithm 1.

  `F`: full objective branching using super local upper bounds (see Algorithm 3).

  `C`: objective branching using a single cone, namely, the nadir point of the local upper bounds dominated by the lower bound set (see Section 4.3).

Table 1: Instances used.

	n[a]	#[b]
AP	36, 49, 64, 121	10
KP	10, 15, 20, 25, 30, 35, 40	30
UFLP	30, 42, 56, 72	10

[a] Number of variables.

[b] Number of instances for each value of n

This leads to six configurations: `B|N`, `B|F`, `B|C`, `D|N`, `D|F` and `D|C`. For an overview of more configurations such as different variable selection rules (Step 5 of the algorithm), the reader is referred to Forget et al. (2020d).

For the comparison with an Objective Space Search (OSS) algorithm, we implemented the approach proposed in Tamby and Vanderpooten (2020), as the authors show that their method outperforms other OSS algorithms from the literature, particularly on knapsack and assignment problems (see the next section). As the LP solver in Bensolve is GLPK, the same MIP solver was used for the OSS. Due to technical constraints in the programming language chosen (Julia), the MIP solver could not be warm started. Warm starting the MIP solver would have led to improved cpu times. However, note that preliminary experiments showed that using CPLEX and warm starting the IPs led to slightly worse computational time. It may be due to the fact that the instances considered do not have a large number of variable. Also note that the same holds for the branch and bound algorithm, which does not use warm starting either when calculating the lower bound set in each node of the branching tree, since this is not possible using Bensolve.

All algorithms and configurations have been tested using a time limit of half an hour (1800 seconds).

5.2 Test instances

A total of 290 instances (see Table 1) have been generated. Three problem classes are considered: the linear assignment problem (AP), the knapsack problem (KP), and the uncapacitated facility location problem (UFLP). These problem classes are widely used in the multi-objective literature for algorithm performance comparisons (see e.g. Stidsen et al. (2014), Gadegaard et al. (2019), Parragh and Tricoire (2019), Tamby and Vanderpooten (2020) and many others). The mathematical programming formulation used for each problem is given in the appendix. The number of variables in each problem class was increased until none of the algorithm configurations was able to compute an optimal solution within a time limit of half an hour (1800 seconds).

The objective function coefficients are generated in the range $[1, 1000]$ on the lower part of a 3D sphere using the R package gMOIP (Nielsen, 2020). Hence, a large number of the coefficient vectors for the variables are non-dominated among each other (91% for the AP and 96% for the KP). This way of generating the objective coefficients was tested against other methods and seems to result in solutions with a large number of non-dominated points (Forget et al., 2020d), implying hard instances. For the UFLP instances the same generation method was used. However, since two different types of costs exist in this case (namely transportation costs and opening costs), a range of $[1, 1000]$ was used for the cost of assigning a customer to a service point, and a range of $[1, 100]$ for the cost of opening a service point. The objective coefficients are all integer.

For the AP the constraints are fixed given the problem size. The same holds for the UFLP when assuming the number of facilities equals the number of customers. For the KP instances, however, the integer coefficients of the constraint are generated randomly in the range $[1, 15]$. The right-hand side is set equal to half of the sum of the coefficients on the left-hand side, rounded down to the next integer. For the KP, three different constraints are generated for each problem size and objective coefficients.

All instances are publicly available at the Multi-Objective Optimization Repository (MOrepo - Nielsen (2017)) in a sub-repository Forget, Nielsen, and Gadegaard (2020a). Moreover, the repository also contains further instances used for preliminary testing (only the hardest instances are kept in this paper). An overview of all results is given in Forget et al. (2020d). The non-dominated sets for each instance are available together with detailed results and plots of some of the instances (Forget, Nielsen, and Gadegaard, 2020b).

5.3 Performance of the different algorithm configurations

First, we rank the configurations with respect to the mean cpu time for all instances. The sequence from best to worst becomes B|C (0%), D|C (15%), B|N (16%), D|N (33%), B|F (50%), D|F (78%), where the increase in percentages compared to the best configuration is given in parentheses. Note that the mean cpu times calculated are in fact a lower bound due to the total run time limit.

Second, a comparison of the different algorithm configurations for a given problem class can be seen in Figure 5. Note that we have increased the variable size for each problem class until the size becomes so large that some instances cannot be solved within the time limit. That is, the number of instances solved before the time limit is naturally below 100%.

From Figure 5, the reader should note the following general observations:

- In general, full objective branching (`B|F` and `D|F`) performs poorly compared to the other configurations. This is especially evident for the AP and the UFLP. For the KP, using full objective branching is still among the worst performing strategies.
- For both the KP and the UFLP, the best objective branching configuration is to use a single cone (`C`), while not using objective branching (`N`) performs the best for the AP.
- In general node selection using breadth-first strategy tends to perform better than depth-first strategy given the objective branching strategy.

These observations will be elaborated upon in the next sections.

Finally, consider the winning algorithm configurations in Figure 6. Note that for the KP and the UFLP, using no objective branching is competitive for some of the smaller instances. Also note that there is no consistently winning configuration. That is, algorithms using different node selection and objective branching strategies during the solution procedure may be useful.

5.4 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 was solved to optimality for all algorithm configurations. Detailed summary statistics are given in Tables 2 and 3.

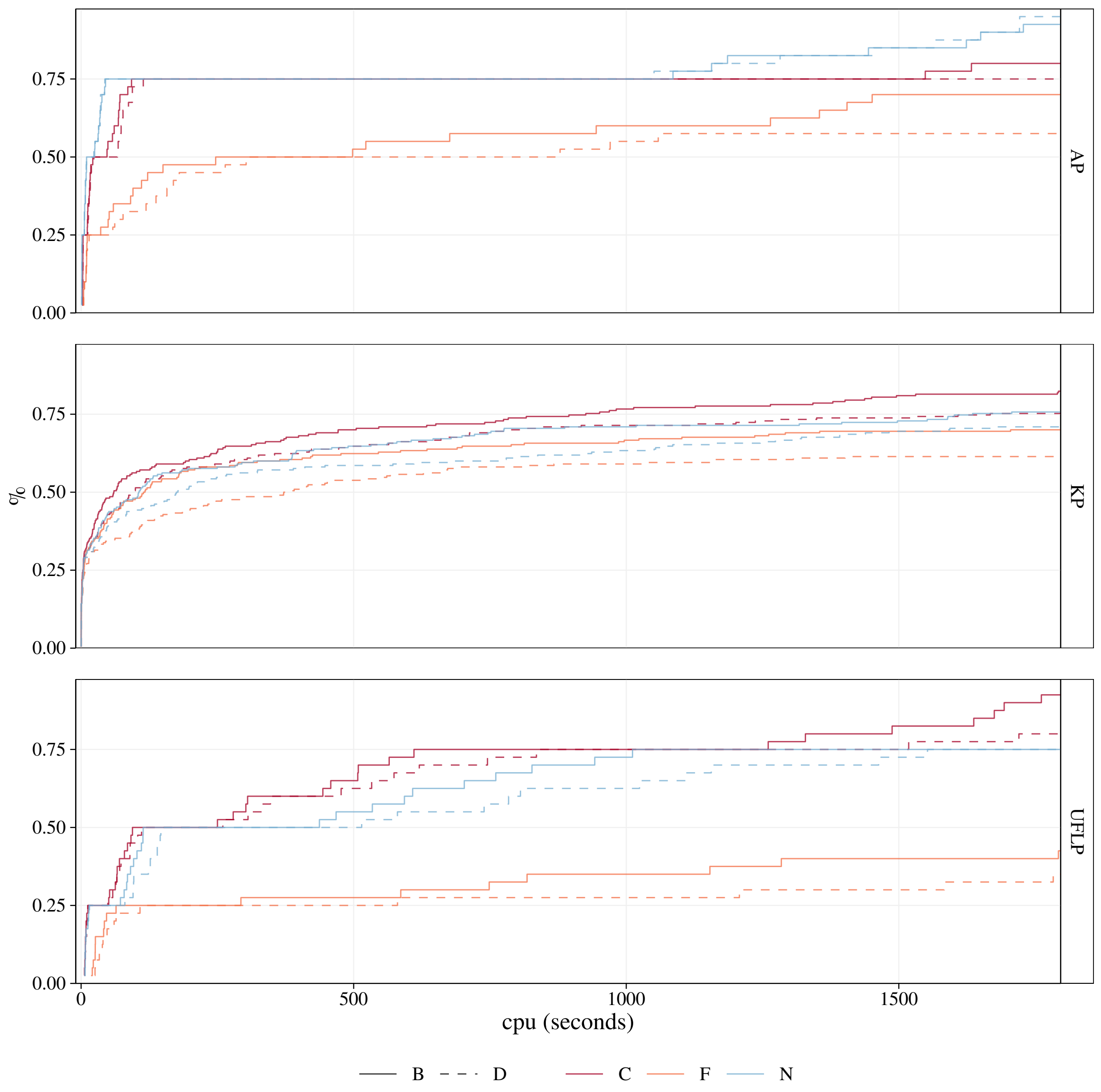


Figure 5: Performance profile: Number of instances in percent solved within a given cpu time. An instance is considered unsolved if the cpu time exceeds 1800 seconds (time limit).

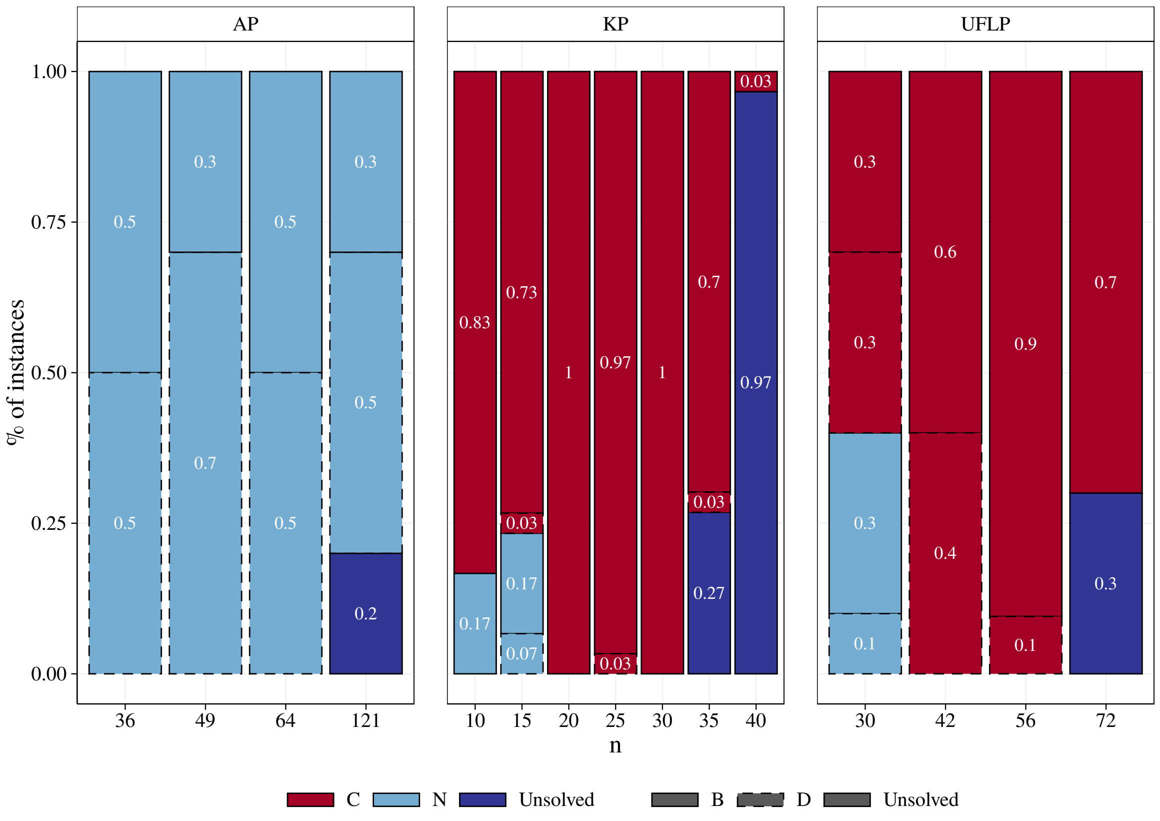


Figure 6: Proportion of winning algorithm configurations.

Table 2: Detailed results for all instances that have been solved to optimality for `B` configurations. The winner among the configurations for a column is highlighted in bold. Summary statistics for each problem class are given.

			`B|C`				`B|F`				`B|N`			
n[a]	#[b]	$\lvert\mathcal{Y}_N\rvert$[c]	cpu[d]	cpuLB[e]	nodes[f]	prune[g]	cpu[d]	cpuLB[e]	nodes[f]	prune[g]	cpu[d]	cpuLB[e]	nodes[f]	prune[g]
AP														
36	10	160 (14/0/86)	2.8 [1.9,3.5]	2.3	1540 [3,18,27]	[83,11,6]	7.8 [3.5,11.2]	2.7	2594 [3,17,26]	[91,5,4]	1.4 [1.1,1.7]	1.5	**807 [6,11,16]**	[0,74,26]
49	10	348 (8/0/92)	14.9 [11.4,21.4]	3.3	4933 [3,25,38]	[82,8,11]	101.2 [35.9,246.8]	3.6	11225 [3,21,37]	[92,3,5]	7.8 [6,10]	2.4	**2573 [7,13,20]**	[0,59,41]
64	3	587 (7/0/93)	51.4 [47.5,57.5]	4.7	11489 [3,29,51]	[78,6,16]	565.5 [498,676.2]	4.4	34192 [3,24,50]	[92,1,7]	28.4 [24.8,35]	3.6	**6182 [8,15,27]**	[0,46,54]
	23	**298 (9/0/91)**	**14.4 [1.9,57.5]**	**3.1**	**4313 [3,23,35]**	**[82,9,9]**	**121.1 [3.5,676.2]**	**3.3**	**10468 [3,20,34]**	**[92,4,5]**	**7.7 [1.1,35]**	**2.2**	**2276 [7,12,19]**	**[0,64,36]**
KP														
10	30	43 (28/0/72)	**0.2 [0.1,0.3]**	0.7	**315 [5,9,11]**	[63,28,8]	0.2 [0.1,0.4]	0.9	350 [5,9,11]	[71,22,6]	0.2 [0.1,0.3]	0.6	336 [5,9,11]	[28,46,26]
15	30	145 (16/0/84)	**2.4 [0.9,5.5]**	1.1	**2238 [7,12,16]**	[67,19,14]	3.7 [1.3,9.2]	1.4	2951 [6,11,16]	[79,12,10]	2.9 [1.3,5]	1.1	2499 [7,13,16]	[12,43,45]
20	30	329 (10/0/90)	**18.2 [2.5,43.8]**	1.5	**10335 [7,16,21]**	[72,12,16]	35.6 [3.3,147.2]	1.9	16899 [6,14,21]	[84,6,11]	26.4 [5,50.6]	1.8	12484 [7,16,21]	[9,43,49]
25	29	550 (8/0/92)	**58.1 [23.5,107.6]**	2.1	**22251 [7,18,26]**	[70,9,20]	111.2 [38.1,218.9]	2.5	39468 [6,16,26]	[84,4,12]	102.1 [46.4,184.8]	3	27918 [7,19,26]	[3,41,56]
30	10	752 (7/0/93)	**182.2 [101.6,250.9]**	3	**42622 [8,20,31]**	[71,7,22]	302.9 [127.2,424.3]	3.3	76974 [6,17,31]	[85,3,12]	317.7 [196.3,395]	5	53069 [8,21,31]	[2,47,52]
	129	**302 (10/0/90)**	**32 [0.1,250.9]**	**1.5**	**11303 [7,14,19]**	**[68,17,15]**	**57.7 [0.1,424.3]**	**1.8**	**19537 [6,13,19]**	**[80,10,10]**	**54.4 [0.1,395]**	**1.9**	**13953 [7,15,19]**	**[12,44,44]**
UFLP														
30	10	325 (14/0/86)	**8.4 [6.2,12.4]**	2.7	**3157 [3,19,26]**	[75,16,9]	33.4 [18.3,64.1]	3.3	6158 [3,17,26]	[89,6,5]	9.4 [5.9,14.5]	2.1	3517 [7,14,21]	[0,65,35]
42	4	900 (8/0/92)	**59 [50.7,70.3]**	4.1	**13026 [4,25,39]**	[76,10,14]	712.8 [293.1,1153.5]	4.1	34729 [4,21,38]	[91,3,6]	90.4 [72.1,113.8]	4.1	16508 [8,17,32]	[0,50,50]
	14	**489 (11/0/89)**	**22.9 [6.2,70.3]**	**3.1**	**5977 [4,21,30]**	**[75,14,10]**	**227.5 [18.3,1153.5]**	**3.5**	**14321 [4,18,30]**	**[90,5,5]**	**32.5 [5.9,113.8]**	**2.7**	**7229 [7,15,24]**	**[0,61,39]**

[a] Number of variables.
[b] Number of instances.
[c] Avg. number of non-dominated points (supported extreme, supported non-extreme and unsupported in percent).
[d] Avg. cpu time (seconds). Square brackets contain the range.
[e] Avg. cpu time (milliseconds) used to find the LB set per node where LB set found.
[f] Avg. number of nodes in the branching tree. Square brackets contain the min, avg. and max depth of leaf nodes.
[g] Percentages of leaf nodes pruned by infeasibility, optimality and dominance.

Table 3: Detailed results for all instances that have been solved to optimality for `D` configurations. The winner among the configurations for a column is highlighted in bold. Summary statistics for each problem class are given.

			`D|C`				`D|F`				`D|N`			
n[a]	#[b]	$\lvert\mathcal{Y}_N\rvert$[c]	cpu[d]	cpuLB[e]	nodes[f]	prune[g]	cpu[d]	cpuLB[e]	nodes[f]	prune[g]	cpu[d]	cpuLB[e]	nodes[f]	prune[g]
AP														
36	10	160 (14/0/86)	3 [2.1,3.7]	2.2	1783 [3,19,27]	[83,12,5]	8.4 [3.5,14.7]	2.7	2640 [3,18,27]	[91,7,2]	**1.4 [1.1,1.6]**	**1.5**	831 [6,11,16]	[0,76,24]
49	10	348 (8/0/92)	16.1 [12.3,23.4]	3.2	6170 [3,26,38]	[83,9,8]	152.7 [58.1,302.7]	3.6	12037 [3,22,38]	[94,3,2]	**7.6 [6,9.8]**	**2.3**	2847 [7,13,21]	[0,62,38]
64	3	587 (7/0/93)	72.5 [68,76.8]	4.5	19421 [3,32,52]	[82,7,11]	969.3 [878.6,1058.7]	4.7	44736 [3,27,51]	[95,2,3]	**28.2 [23.4,35.4]**	**3.4**	7166 [8,15,27]	[0,50,50]
	23	**298 (9/0/91)**	**17.8 [2.1,76.8]**	**2.9**	**5991 [3,24,35]**	**[83,10,7]**	**196.5 [3.5,1058.7]**	**3.3**	**12216 [3,21,35]**	**[93,5,2]**	**7.6 [1.1,35.4]**	**2.1**	**2534 [7,12,20]**	**[0,66,34]**
KP														
10	30	43 (28/0/72)	0.2 [0.1,0.4]	0.7	371 [5,9,11]	[64,32,4]	0.3 [0.2,0.7]	0.8	404 [5,9,11]	[71,26,3]	0.2 [0.1,0.3]	**0.6**	393 [5,9,11]	[33,49,18]
15	30	145 (16/0/84)	3.2 [1.5,7.1]	1	3117 [7,13,16]	[70,21,9]	6.7 [2,19]	1.3	3910 [6,12,16]	[81,14,5]	3.4 [1.9,5.7]	**1**	3540 [7,13,16]	[19,46,35]
20	30	329 (10/0/90)	28.5 [3.2,71.8]	**1.4**	18618 [7,16,21]	[75,14,11]	103.5 [4.4,413.6]	1.8	27738 [6,15,21]	[88,7,5]	39.3 [6.7,83.6]	1.5	23616 [7,17,21]	[14,43,43]
25	29	550 (8/0/92)	98.8 [41.6,184.5]	**1.8**	50927 [8,19,26]	[75,10,15]	366.2 [89,867.4]	2.3	81458 [6,17,26]	[90,5,6]	182.2 [82.2,323.8]	2.4	68127 [7,20,26]	[6,44,50]
30	10	752 (7/0/93)	247.4 [147,325.3]	**2.4**	102150 [8,21,31]	[76,9,15]	896.6 [185.9,1401.7]	2.8	164866 [5,19,31]	[90,4,6]	621.1 [323.1,837.5]	4	138892 [7,23,31]	[4,48,48]
	129	**302 (10/0/90)**	**48.8 [0.1,325.3]**	**1.3**	**24508 [7,15,19]**	**[71,19,10]**	**177.5 [0.2,1401.7]**	**1.6**	**38546 [6,14,19]**	**[83,12,5]**	**99.1 [0.1,837.5]**	**1.6**	**32489 [7,15,19]**	**[17,46,37]**
UFLP														
30	10	325 (14/0/86)	8.9 [6.1,12.3]	2.6	3868 [3,20,26]	[75,18,7]	47.2 [25.3,108]	3.1	6758 [3,17,26]	[90,8,3]	9.4 [6.8,15.3]	**1.9**	4270 [7,14,22]	[0,70,30]
42	4	900 (8/0/92)	61.1 [48.9,73.2]	3.8	17300 [4,25,39]	[77,12,10]	1288.7 [580.4,1783.3]	4.1	39028 [4,22,39]	[93,4,3]	110 [80.3,139.7]	**3.6**	25250 [8,18,32]	[0,55,45]
	14	**489 (11/0/89)**	**23.8 [6.1,73.2]**	**3**	**7705 [4,21,30]**	**[76,17,8]**	**401.9 [25.3,1783.3]**	**3.4**	**15978 [4,19,30]**	**[91,7,3]**	**38.1 [6.8,139.7]**	**2.4**	**10264 [7,15,25]**	**[0,66,34]**

[a] Number of variables.
[b] Number of instances.
[c] Avg. number of non-dominated points (supported extreme, supported non-extreme and unsupported in percent).
[d] Avg. cpu time (seconds). Square brackets contain the range.
[e] Avg. cpu time (milliseconds) used to find the LB set per node where LB set found.
[f] Avg. number of nodes in the branching tree. Square brackets contain the min, avg. and max depth of leaf nodes.
[g] Percentages of leaf nodes pruned by infeasibility, optimality and dominance.

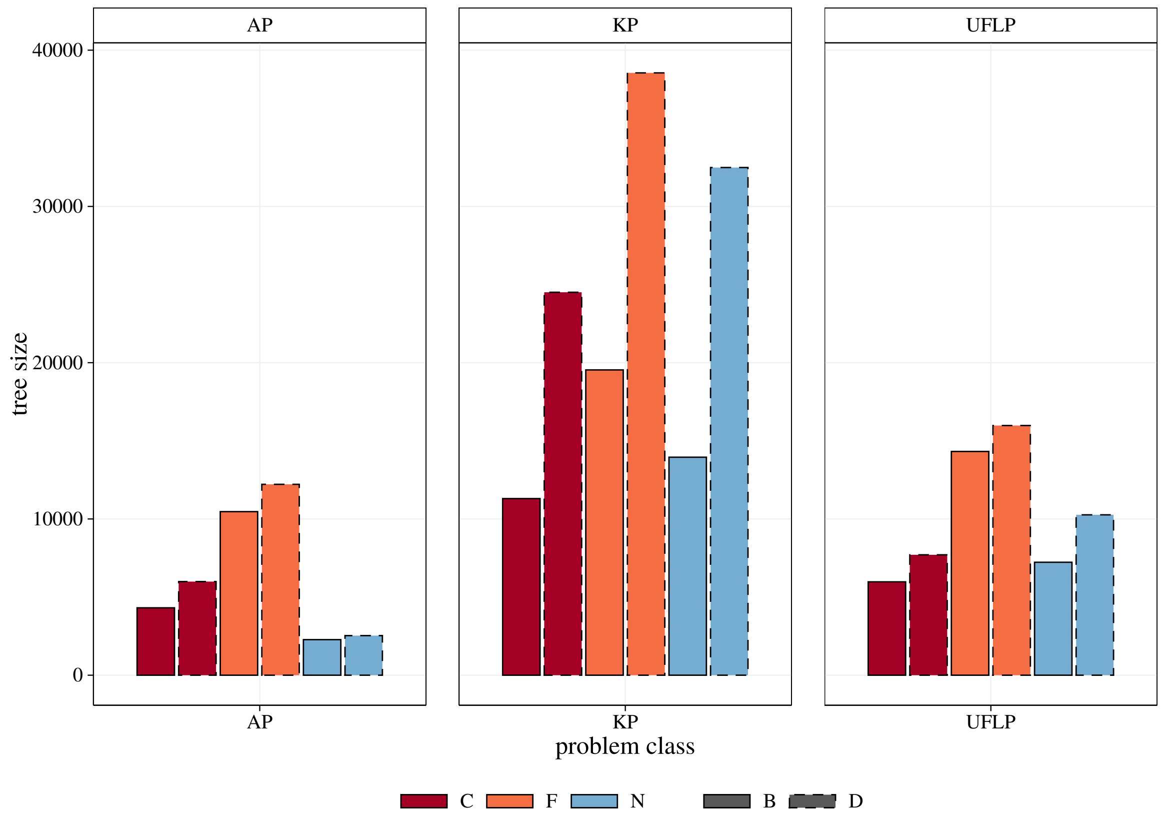


Figure 7: Average branching tree size.

Figure 5 illustrates that using full objective branching (`F`) is not efficient with respect to cpu time. As explained in Section 4.3, full objective branching creates more nodes (on average 3.1 nodes) in the B&B tree as it partitions the objective space into smaller regions. This leads to longer computation times, in particular since computing the lower bound set of each node is computationally very expensive. Indeed, this can be seen in Figure 7 where `F`-configurations produce more nodes in the search tree compared to `C`- and `N`-configurations given the same node selection rule. This observation makes sense and was expected, as the purpose of objective branching described in Algorithm 3 is to produce more subproblems and thus get wider trees, with the expectation that these subproblems will be easier to solve and that the tree will have a smaller depth. However, the branching nodes produced using full objective branching take longer to process, on average. This can also be seen in Table 2 and 3 if we compare the time used to find lower bound sets for configurations `F` and `C`. That is, the subproblems generated in the `F`-configurations tend to be harder to solve on average due to a more complicated set of constraints. Actually, as the trees are generally wider when full objective branching is applied, more nodes with fewer variables fixed to 0 or 1 (i.e., close to the root node) are generated. Unfortunately, limiting the B&B algorithm to smaller areas of the objective space using full objective branching does not compensate for the larger number of nodes with fewer fixed variables.

To make full objective branching competitive, the cpu time for calculating the lower bound set needs to be reduced. That is, we need to compute the lower bound set faster than what is currently done using BenSolve. This would lead to large reductions in the overall computation time as most of the computational time is spent on computing the lower bound sets (approx. 66%). This would, of course, benefit all the configurations, but especially the `F` configurations, as they generate more nodes. Some ideas for reducing the cpu time could be to

- implement a way to warm start the LP solver for each subproblem,
- introduce an updating procedure of the lower bound set, instead of computing it from scratch at each node,
- use further pruning rules that discard subproblems before calculating the lower bound set,
- use only few weighted sum problems to compute (rough) lower bound sets.

Updating and pruning procedures may include using the local information available in the subproblems with respect to the region in the objective space defined by a branching node. In Adelgren and Gupte (2019) it was shown that presolve techniques, contrary to the case of single-objective optimization, have a great effect deeper in the search tree. Hence, one could imagine that the solutions are more likely to be similar in a specific part of the objective space and thus, by localizing the branch and bound, presolving the nodes or introducing cuts may have a significant impact.

Except for the AP, objective branching using a single cone (`C`) performs better than the `N`-configurations in both cpu time and number of tree nodes. That is, objective branching is competitive. The intuitive explanation of the efficiency of `C`-configurations for the KP and the UFLP is that for a given node η, the `C`-configurations will focus on a specific part of the objective space (a cone defined by $d^N(\eta)$) for the same number of subproblems created (two, in the decision space). Thus, the lower bound sets tend to be less complex (some areas of the objective space are discarded before computation, while it is fully considered in the `N`-version, without generating additional nodes as in the `F`-version), which possibly means spending less time in the most expensive part of the B&B algorithm (computing the lower bound sets). Second, the way nodes are fathomed is highly influenced by the choice of objective branching strategy. Indeed, as it can be seen in Tables 2 and 3, in the `C`-configurations, the proportion of nodes fathomed by infeasibility tends to be much higher than for the `N`-configurations. This is due to the fact that additional constraints added in the `C`-configurations increase the likelyhood that subproblems are infeasible. Moreover, fathoming a node by infeasibility is faster than fathoming a node by optimality or by dominance. To fathom by dominance or optimality necessitates computing the lower bound set whereas testing the feasibility of the subproblem is the first thing done when a node is being explored.

The case of the AP is different from the KP and the UFLP. This is not surprising as the constraint matrix of the AP is totally unimodular, implying that the extreme points of the polytope defined by the constraints are integral. Hence, all the extreme points of the lower bound sets correspond to feasible solutions in the `N`-configurations. This implies that, in Step 5, the branching rule will exploit this and thus, always branch on the variable that most often differs. As new constraints are added in the `C`-configurations, however, the extreme points are not necessarily integer and the branching rule will choose to branch on the most often fractional variable. This difference in the branching rule may have a significant impact and explain why the `N`-configurations are better.

5.5 Node selection: breadth-first search versus depth-first search

In Figure 5, it was shown that the node selection using a breadth-first strategy tends to perform better than a depth-first strategy given an objective branching rule. This observation is in contrast to the prevalent use of depth-first in multi-objective branch and bound algorithms (Przybylski and Gandibleux, 2017).

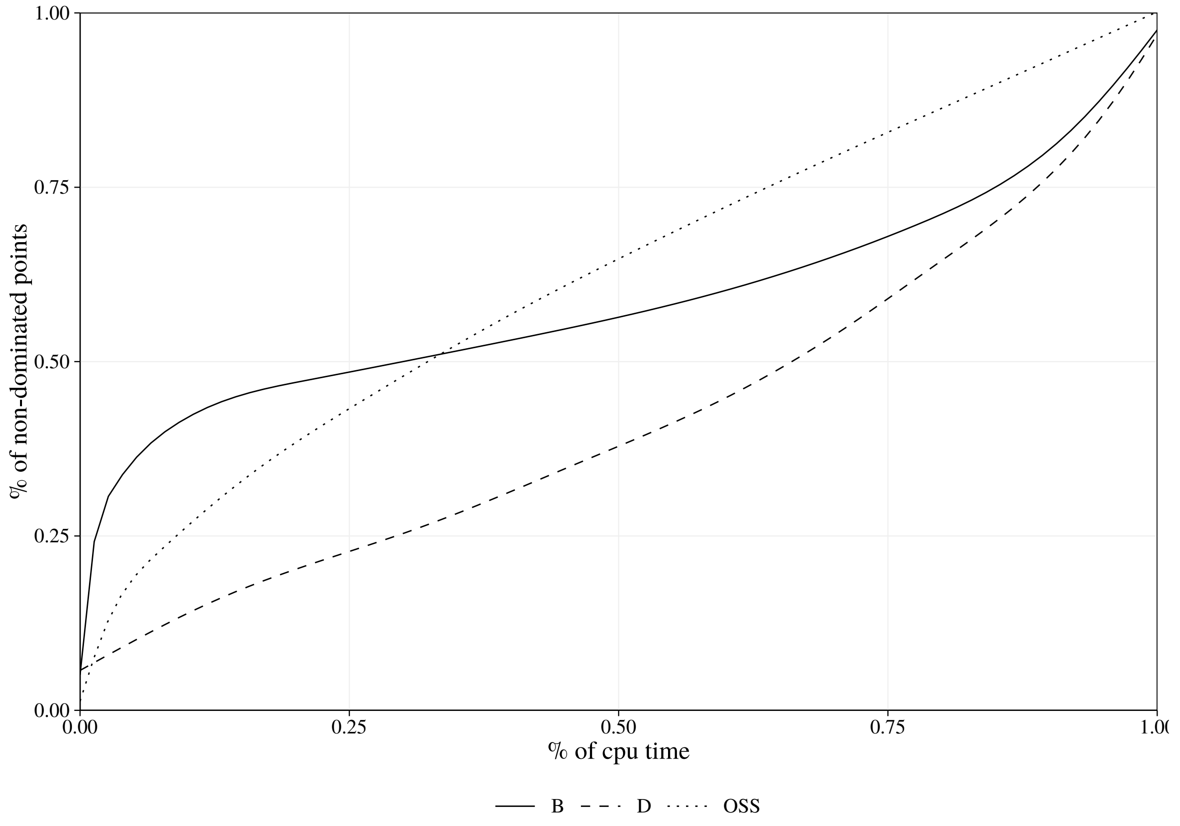


Figure 8: Smoothed curves of relative number of non-dominated points compared to relative cpu.

A possible explanation is that with a depth-first strategy, the branch and bound algorithm tends to focus on specific parts of the objective space, one after the other. However, using a breadth-first strategy offers a better coverage of all the objective space at an earlier stage and thus, yields better upper bound sets earlier. This explanation is confirmed if we take a look at Figure 8 where all the instances solved to optimality with no objective branching (`N`) or using a cone (`C`) have been grouped based on the node selection strategy. Observe that using a breadth-first strategy (`B`) in general finds an approximation of the non-dominated set relatively faster, i.e., we obtain a better upper bound set. Since the figure reports relative numbers, it could be the case that a depth-first strategy would still be better if the cpu time is much smaller than a breadth-first strategy. Note, however, that using configurations based on a breadth-first node selection strategy in general find an optimal solution faster than a depth-first node selection strategy (see Figure 5). It also implies that breadth-first search may not perform as well compared to depth-first search in case the upper bound set is initialized with a good heuristic, which would thus yield a good upper bound set from the start, regardless of the node selection strategy used.

In addition, one can observe in Figure 8 that the curve that represents the number of non-dominated points found for breadth-first strategies tends to increase very quickly in the beginning and slowly flattens in the middle, and finally increases steeply again towards the end. In the beginning, the first layers of the tree are explored. As the problems in these layers are less constrained and the upper bound set is relatively sparsely populated, many new non-dominated points are found. When the B&B algorithm reaches deeper layers, towards the middle, the problems become more constrained, which generates less new non-dominated points. Towards the end, the deepest parts of the branch and bound tree are explored. The problems here are so constrained that they become much easier to solve (at most one feasible solution at layer $n + 1$, the deepest possible layer in the tree) and thus, are processed much faster. In addition to that, the solutions found deep in the tree are more likely to be integer and thus to generate new non-dominated points. This leads to the acceleration of the number of non-dominated points found near the end of the algorithm. This observation also suggests that it may be interesting to combine breadth- and depth-first strategies to quickly reach the parts of the tree that generate many non-dominated points.

5.6 Comparing the B&B algorithm with an objective space search algorithm

The performance of the B&B algorithm compared to the Objective Space Search (OSS) algorithm is shown in Figure 9. In general, the OSS algorithm performs better than the B&B algorithm (on average 336 seconds faster). For the KP, the OSS algorithm performs much better than the current best branch and bound configuration. It is the case for UFLP as well, but the gap is smaller. Regarding AP, the gap is even smaller. A possible explanation is that again, as constraints are added to the initial problem, the total unimodularity of the constraint matrix is lost, which leads to harder problems (harder than a single-objective AP) for the MIP solver, while the B&B algorithm without objective branching benefits from this property.

We note that it doesn't come as a surprise that the OSS algorithm is highly competitive and often outperforms multiobjective B&B algorithms since it benefits from the power of single-objective MIP solvers, which have been 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 analyse the concept of objective branching in a B&B algorithm (that has proved to be successful for bi-objective problems) in the tri-objective case. These continuing efforts are the basis for an advancement of the development of promising methods that hybridize between decision space and objective space search.

Note also that the B&B algorithm maintains a single branching tree in contrast to the OSS algorithm that creates a new branching tree for each call to the MIP solver. Hence, if the average computation in each node of the B&B algorithm can be reduced, then B&B algorithms may outperform OSS. The average number of nodes in the branching tree compared to the number of MIPs solved is approximately 16 times larger. That is, processing a node in the branching tree should be approximately 16 times lower on average for the B&B algorithm to be competitive. Processing a node is currently by a factor of 3.8 lower for the B&B algorithm, which means that in order to be competitive, processing a node should take 5.4 times less time than it does now.

We believe that closing this gap is possible with further research in pruning nodes and finding lower bound sets faster (i.e., processing nodes faster) or by considering alternative node and variable selection rules (i.e., reducing the size of the tree and thus, decreasing this factor of 16). Also, the B&B algorithm uses no cut-generation, which is regularly used by the single objective MIP solvers. Thus, generating cuts, and effectively extending the B&B algorithm to a branch and cut algorithm could lead to reduced tree sizes.

6 Conclusion

The purpose of this paper was to propose a multi-objective Branch-and-Bound (B&B) framework, extend the principle of objective branching to the multi-objective case, and to study its impact on the algorithm.

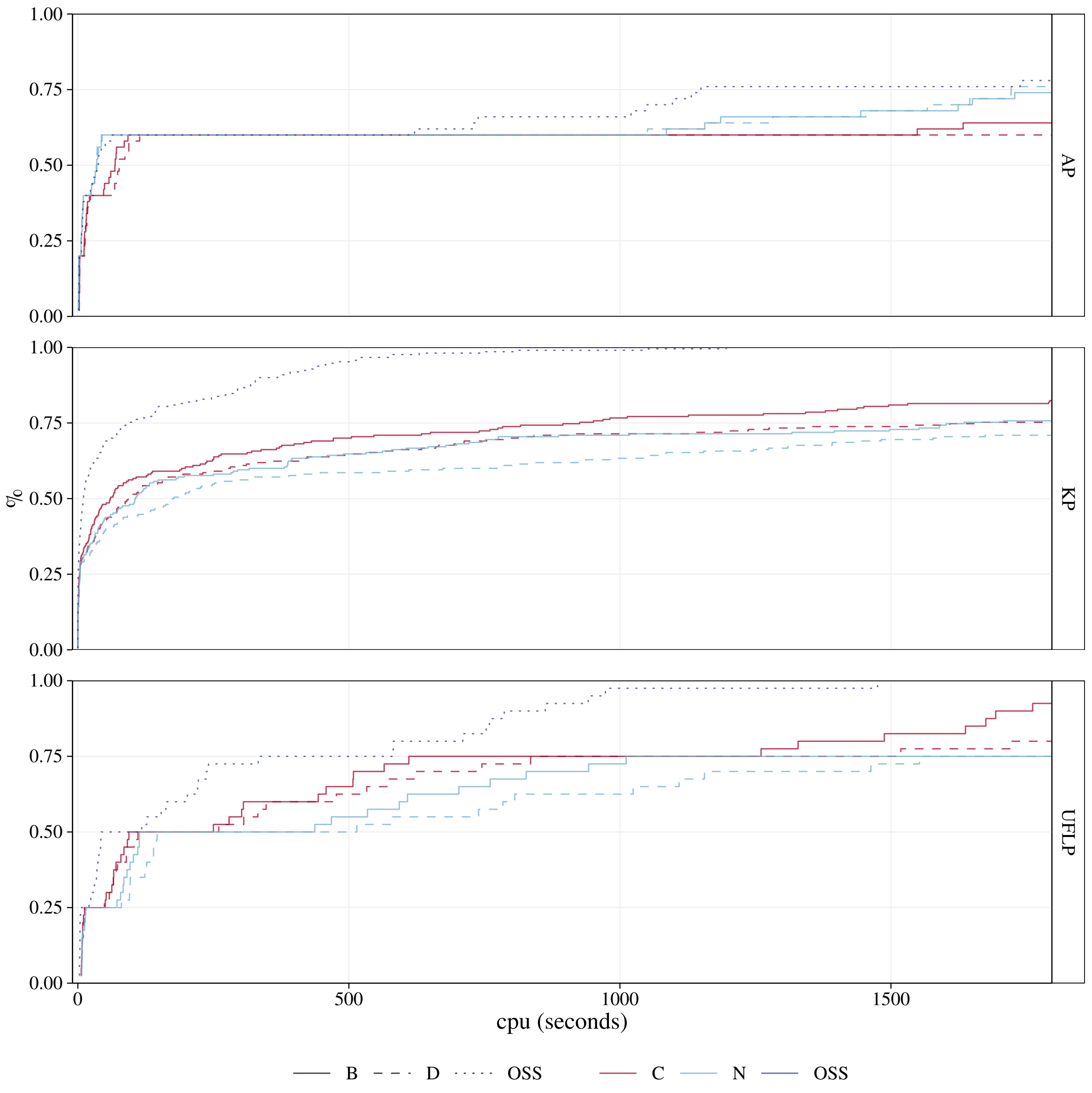


Figure 9: Performance profile: Number of instances in percent solved within a given cpu time. An instance is considered as unsolved if the cpu time exceeds 1800 seconds (time limit).

First, a B&B algorithm capable of solving a MOCO with any number of objectives was proposed in Section 3. It was based on the use of linear relaxations for the computation of lower bound sets and came with a dominance test inspired by existing literature. We then highlighted a set of difficulties when extending objective branching from the bi-objective case to the multi-objective case. In order to overcome these difficulties, a number of desirable properties of objective branching was exposed, and the concept of a 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.

The experiments in Section 5 showed that except for the special case of the Assignment Problem, cone objective branching performs the best. Indeed, this version of objective branching processes each node faster while limiting the number of nodes generated at the same time, which leads to better computational times.

This contrasts with the current B&B framework where applying full objective branching as described in Algorithm 3 generates more nodes. This additional number of nodes is currently not processed fast enough, which leads to worse computational times compared to not using objective branching. However, the promising results indicate that by exploiting local information from the subproblems to a larger extent and by processing the nodes faster, an overall robust and fast decision space search algorithm is within reach. The experiments have shown that in this framework, the most expensive part in terms of cpu time is the computation of the lower bound sets. Hence, improving this part in future research may be beneficial both for the B&B algorithm in general, and for full objective branching in particular.

References

Nathan Adelgren and Akshay Gupte. Branch-and-bound for biobjective mixed-integer linear programming. Submitted to INFORMS Journal on Computing, 2019. URL `http://www.optimization-online.org/DB_FILE/2016/10/5676.pdf`.

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

T.M. Benson. An outer approximation algorithm for generating 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.

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.

M. Ehrgott. *Multicriteria Optimization*. Springer Berlin, Heidelberg, 2nd edition, 2005. ISBN 3540213988.

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, L. R. Nielsen, and S. L. Gadegaard. Instances and results for tri-objective branch and bound (morepo-forget20), 2020a. URL `https://github.com/MCDMSociety/MOrepo-Forget20`.

N. Forget, L. R. Nielsen, and S. L. Gadegaard. Detailed results for instances (morepo-forget20), 2020b. URL `https://mcdmsociety.github.io/MOrepo-Forget20/report_details.html`.

N. Forget, L. R. Nielsen, and S. L. Gadegaard. Examples of lower and upper bound sets (morepo-forget20), 2020c. URL `https://mcdmsociety.github.io/MOrepo-Forget20/report_interactive.html`.

N. Forget, L.R. Nielsen, and S.L Gadegaard. Computational results (all instances). Technical report, Aarhus University, 2020d. URL `https://mcdmsociety.github.io/MOrepo-Forget20/report.html`. Results for all the instances at the repository MOrepo-Forget20.

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*, 2019.

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.

L.R. Nielsen. Multi-objective optimization repository (morepo), 2017. URL `https://github.com/MCDMSociety/MOrepo`.

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

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*, 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. GonzÃ¡lez. 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.

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*, 2020. doi: 10.1287/ijoc.2020.0953. Ahead of print.

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.