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

Nicolas Forget†, Sune Lauth Gadegaard, Lars Relund Nielsen

Department of Economics and Business Economics, School of Business and Social Sciences, Aarhus University, Fuglesangs Allé 4, DK-8210 Aarhus V, Denmark

August, 2021

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

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

## 1 Introduction

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

*Preprint of N. Forget, S.L. Gadegaard and L.R. Nielsen, Warm-starting lower bound set computations for branch-and-bound algorithms for multi objective integer linear programs in European Journal of Operational Research (2022), doi:10.1016/j.ejor.2022.01.047

†Corresponding author (nforget@econ.au.dk).

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

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

Over the past decades, various methodologies have been proposed in the literature to solve MOILPs. These methodologies can be roughly divided into two main categories: *Objective Space Search* (*OSS*) algorithms and *Decision Space Search* (*DSS*) algorithms. The principle of an OSS algorithm is to search the objective space by solving a series of single-objective problems, obtained by *scalarizing* the objective functions (Ehrgott, 2005). Hence, the power of single-objective solvers can be used to generate the optimal set of solutions (see Section 2 for a formal definition). Consequently, much attention has been paid to OSS methods over the years (see e.g. Ulungu and Teghem (1995); Visée, Teghem, Pirlot, and Ulungu (1998); Sylva and Crema (2004); Ozlen, Burton, and MacRae (2014); Kirlik and Sayın (2014); Boland, Charkhgard, and Savelsbergh (2017); Boland and Savelsbergh (2016); Tamby and Vanderpooten (2021)).

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

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

in Florios, Mavrotas, and Diakoulaki (2010) for the multi-objective case.

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

Vincent et al. (2013) showed that for bi-objective problems with computationally demanding single-objective versions, the linear relaxation is often preferable in terms of computation times, even though it leads to larger branch-and-bound trees. They also showed that the linear relaxation is preferable to the ideal point and to the ideal point of the linear relaxation. Moreover, the authors proposed an extension of their branch-and-bound framework to *Bi-Objective Mixed-Integer Problems* (BOMIP) along with an alternative dominance test. The class of BOMIP was also studied by Belotti, Soylu, and Wiecek (2013), and improved in Belotti, Soylu, and Wiecek (2016) who developed stronger fathoming rules. Finally, Adelgren and Gupte (2021) provided an extensive study on BOMIP and incorporated the recent knowledge of DSS algorithms in their framework.

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

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

It appears that the branch-and-bound frameworks developed over the last decade are competitive when

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

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

The remainder of this paper is organized as follows: In Section 2 we present the preliminaries and in Section 3 we present a generic branch-and-bound framework for MOILPs. Section 4 describes how lower bound sets can be generated using a Benson-like algorithm and how such an algorithm can be modified so warm-starting becomes possible. In Section 5 we conduct an extensive computational study, and finally Section 6 concludes the paper.

## 2 Preliminaries

In multi-objective optimization, not only one but several conflicting objectives are considered simultaneously, and hence it is most often impossible to find one solution optimizing all objectives at the same time. Therefore, it is necessary to introduce operators for the comparison of points and sets. Given $y^1, y^2 \in \mathbb{R}^p$, the point $y^1$

*weakly dominates* $y^2$ ($y^1 \leqq y^2$) if $y^1_k \leq y^2_k, \forall k \in \{1, ..., p\}$. Moreover, we say that $y^1$ *dominates* $y^2$ ($y^1 \leqslant y^2$) if $y^1 \leqq y^2$ and $y^1 \neq y^2$. These dominance relations can be extended to sets of points as follows: Let $\mathcal{A}, \mathcal{B} \subseteq \mathbb{R}^p$, we say that $\mathcal{A}$ *dominates* $\mathcal{B}$ if for all $b \in \mathcal{B}$, there exists $a \in \mathcal{A}$ such that $a \leqslant b$. Furthermore, a subset $\mathcal{A} \subset \mathbb{R}^p$ is said to be *stable* if for any $a, a' \in \mathcal{A}$, $a \nleqslant a'$.

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

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

where $\mathcal{X} = \{x \in \mathbb{N}^n_0 \mid Ax \geqq b\}$ is the *feasible set* in the *decision space*. We assume that $\mathcal{X}$ is bounded (if this is not the case, it will be detected by our algorithm). The matrix $A \in \mathbb{R}^{m \times n}$ defines the coefficients of the $m$ constraints with right-hand side $b \in \mathbb{Z}^m$. The $p$ linear objectives are defined using the matrix $C \in \mathbb{Z}^{p \times n}$ of *objective function coefficients*. The corresponding *set of feasible objective vectors* in the *objective space* is $\mathcal{Y} = \{z(x) \mid x \in \mathcal{X}\} := C\mathcal{X}$.

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

## 2.1 Polyhedral theory

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

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

A *vertex* of $\mathcal{P}$ is a face of dimension zero. The vector $r \in \mathbb{R}^p$ is a *ray* of $\mathcal{P}$ if $x + \lambda r \in \mathcal{P}$ for all $x \in \mathcal{P}$ and $\lambda \geq 0$. A ray $r$ of $\mathcal{P}$ is said to be *extreme* if $r = \lambda_1 r^1 + \lambda_2 r^2$ where $r^1$ and $r^2$ are rays of $\mathcal{P}$ and $\lambda_1, \lambda_2 > 0$ implies that $r^1 = \lambda r^2$ for some $\lambda > 0$. A facet of a polyhedron $\mathcal{P}$ can be described using a finite set of vertices $\mathcal{V}_F$ and extreme rays $\mathcal{R}_F$ satisfying $\mathcal{F} = \text{conv}(\mathcal{V}_F) + \{\sum_{r \in \mathcal{R}_F} \lambda_r r, \lambda \geqq 0\}$ (convex hull of vertices and rays). Since the boundary of a polyhedron consists of a finite set of facets, a *vertex-ray representation* of polytope $\mathcal{P}$ is $\mathcal{P}_V = (\mathcal{V}_P, \mathcal{R}_P)$ satisfying $\mathcal{P} = \text{conv}(\mathcal{V}_P) + \{\sum_{r \in \mathcal{R}_P} \lambda_r r, \lambda \geqq 0\}$. In general, if we use a representation of $\mathcal{P}$ using $(\mathcal{P}_H, \mathcal{P}_V)$, the sets $\mathcal{P}_H$ and $\mathcal{P}_V$ are linked together using e.g. an adjacency list so it is known which

vertices are adjacent, which vertices and rays belong to which facets, and vice versa. Note that $\mathcal{P}$ is a polytope if and only if no extreme ray exists, i.e. $\mathcal{R}_F = \emptyset$ and rays can be dropped from $\mathcal{P}_V$.

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

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

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

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

## 2.2 Bound sets

Given a set of points $\mathcal{S} \subseteq \mathbb{R}^p$, it is possible to define lower and upper bound sets for $\mathcal{S}_N$. For this purpose, the definition from Ehrgott and Gandibleux (2007), recalled in Definition 1, will be used. A subset $\mathcal{S}$ is $\mathbb{R}^p_{\geqq}$-closed if $\mathcal{S} + \mathbb{R}^p_{\geqq}$ is closed, and $\mathbb{R}^p_{\geqq}$-bounded if there exists $y \in \mathbb{R}^p$ such that $\mathcal{S} \subset \{y\} + \mathbb{R}^p_{\geqq}$.

**Definition 1.** *(Ehrgott and Gandibleux, 2007) Let $\mathcal{S} \subseteq \mathbb{R}^p$ be a set.*

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

Ehrgott and Gandibleux (2007) showed that the singleton $\{y^I\}$, denoted the *ideal point* and defined by $y^I_k = \min_{y \in \mathcal{Y}}\{y_k\}$, is a valid lower bound set for $\mathcal{Y}_N$. The same holds for the non-dominated set of the linear relaxation P$^{\text{LP}}$ of P. Moreover, the *anti-ideal point* $\{y^{AI}\}$, defined as $y^{AI}_k = \max_{y \in \mathcal{Y}}\{y_k\}$, yields a valid upper bound set for $\mathcal{Y}_N$. A variant of the anti-ideal point is the *nadir point*, defined as $y^N_k = \max_{y \in \mathcal{Y}_N}\{y_k\}$. The authors also showed that, in the context of a branch-and-bound algorithm, the *incumbent set*, which is the current stable set of solutions found at any point during the algorithm, is a valid upper bound set for $\mathcal{Y}_N$.

An upper bound set $\mathcal{U}$ can alternatively be described in terms of its corresponding set of *local upper bounds* $\mathcal{N}(\mathcal{U})$ (sometimes also referred to as *local nadir points*). This concept was formally defined by Klamroth et al. (2015), and their definition is recalled in Definition 2. Let $C(u) = u - \mathbb{R}^p_{\geqq} := \{y \in \mathbb{R}^p \mid y \leqq u\}$ be the *search cone* of $u \in \mathbb{R}^p$.

**Algorithm 1** Branch-and-bound algorithm for a MOILP.

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

**Definition 2.** *(Klamroth et al., 2015) The set of local upper bounds of $\mathcal{U}$, $\mathcal{N}(\mathcal{U})$, is a set that satisfies*

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

# 3 A branch-and-bound framework for MOILP

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

A general description of a *multi-objective branch-and-bound* (*MOBB*) framework for solving problem P is given in Algorithm 1. The algorithm manages a branching tree, $\mathcal{T}$, where each node $\eta$ contains a sub-problem of P. At each node $\eta$, the sub-problem contained in $\eta$ is denoted by $P(\eta)$, and its feasible set and set of feasible objective vectors are $\mathcal{X}(\eta)$ and $\mathcal{Y}(\eta)$ respectively. Similarly, the set of non-dominated points of $P(\eta)$ is given by $\mathcal{Y}_N(\eta)$. We define analogously $\mathcal{X}^{LP}(\eta)$, $\mathcal{Y}^{LP}(\eta)$ and $\mathcal{Y}_N^{LP}(\eta)$ for the linear relaxation $P^{LP}(\eta)$ of $P(\eta)$.

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

Implementations of a MOBB algorithm may differ in the *node selection rule* (line 3), in the way the lower bound set is calculated (line 4), and in how the upper bound set is updated (line 5). Moreover, different *pruning*

*rules* may be used to remove a node from the candidate set (line 6). Finally, different *variable selection rules* may be used to split a *father node* into a set of *child nodes* (lines 7-8).

As node selection rule we use the so-called *breadth first* search strategy, which follows a FIFO principle, meaning that we always chose the unprocessed node that was created first. We use the non-dominated set $\mathcal{Y}_N^{LP}(\eta)$ of the linear relaxation $P^{LP}(\eta)$ as a lower bound set in each node. We use a revisited state-of-the-art version of Benson's outer approximation algorithm using warm-starting (see Section 4) where the polyhedron $\mathcal{Y}_N^{LP}(\eta) + \mathbb{R}^p_{\leqq}$ is found with both a half-space and vertex and ray representation. Since an integer-feasible solution to $P^{LP}(\eta)$ is feasible for $P(\eta)$, the upper bound set can be updated using the vertex representation of the lower bound set $\mathcal{Y}_N^{LP}(\eta)$ by adding vertices corresponding to integer solutions to $\mathcal{U}$ and removing any dominated points.

Different rules can be used to prune a node as well. If $P^{LP}(\eta)$ is not feasible (i.e. $\mathcal{X}^{LP}(\eta) = \emptyset$), then $P(\eta)$ is not feasible either since $\mathcal{X}(\eta) \subseteq \mathcal{X}^{LP}(\eta) = \emptyset$, and hence the node is *pruned by infeasibility*. In the case where $\mathcal{Y}_N^{LP}(\eta) + \mathbb{R}^p_{\leqq}$ contains a single vertex $y$ with a feasible pre-image, the node can be *pruned by optimality* since all points in $\mathcal{Y}(\eta)$ are weakly dominated by $y$. Finally, if $\mathcal{Y}_N^{LP}(\eta) + \mathbb{R}^p_{\leqq}$ is dominated by $\mathcal{U}$, the node can be *pruned by dominance*. In practice, the latter rule is checked by applying the methodology used for the bi-objective case in Sourd and Spanjaard (2008) and in Gadegaard et al. (2019), since it extends naturally to the multi-objective case. This is recalled in Lemma 1.

**Lemma 1.** *Let $\mathcal{U}$ be an upper bound set for $\mathcal{Y}_N$. The node $\eta$ can be pruned by dominance if for each $u \in \mathcal{N}(\mathcal{U})$, $u \notin \mathcal{Y}_N^{LP}(\eta) + \mathbb{R}^p_{\leqq}$ holds true.*

*Proof.* First, for any non-dominated point $y \in \mathcal{Y}_N$ of the initial problem $P$, there exists at least one $u \in \mathcal{N}(\mathcal{U})$ such that $y \leqq u$. Indeed, from Definition 1 and Definition 2, we have that $\mathcal{Y}_N \subset \text{cl}[\mathbb{R}^p \backslash (\mathcal{U} + \mathbb{R}^p_{\leqq})]$ and $\text{cl}[\mathbb{R}^p \backslash (\mathcal{U} + \mathbb{R}^p_{\leqq})] = \bigcup_{u \in \mathcal{N}(\mathcal{U})} C(u)$. Thus, $\mathcal{Y}_N \subset \bigcup_{u \in \mathcal{N}(\mathcal{U})} C(u)$. This implies that for each $y \in \mathcal{Y}_N$, there exists $u \in \mathcal{N}(\mathcal{U})$ such that $y \in C(u)$ and consequently, by the definition of $C(u)$, there exists $u \in \mathcal{N}(\mathcal{U})$ such that $y \leqq u$. Furthermore, we know that there is no $u \in \mathcal{N}(\mathcal{U})$ such that $u \in \mathcal{Y}_N^{LP}(\eta) + \mathbb{R}^p_{\leqq}$. It is not possible that $y \in \mathcal{Y}_N^{LP}(\eta) + \mathbb{R}^p_{\leqq}$ if $u \notin \mathcal{Y}_N^{LP}(\eta) + \mathbb{R}^p_{\leqq}$, because by construction of $\mathbb{R}^p_{\leqq}$, for any set $\mathcal{S} \subset \mathbb{R}^p$ and for any $s \in \mathcal{S} + \mathbb{R}^p_{\leqq}$, $\{s\} + \mathbb{R}^p_{\leqq} \subseteq \mathcal{S} + \mathbb{R}^p_{\leqq}$. Hence, since $y \leqq u$, we have that $u \in \{y\} + \mathbb{R}^p_{\leqq}$ and thus, $u \in \mathcal{Y}_N^{LP}(\eta) + \mathbb{R}^p_{\leqq}$, which is a contradiction. This implies that necessarily, $y \notin \mathcal{Y}_N^{LP}(\eta) + \mathbb{R}^p_{\leqq}$, and as a result, no new non-dominated point can be found in sub-problem $P(\eta)$. □

If the node cannot be pruned, $P(\eta)$ is divided into easier sub-problems. Like in the single-objective case, two disjoint sub-problems are traditionally created by using a variable selection rule to choose a variable $x_i$ and imposing bounds on this variable. Usually, one sub-problem will be generated with the feasible set $\{x \in \mathcal{X}(\eta) \mid x_i \leq z\}$, and the other with the feasible set $\{x \in \mathcal{X}(\eta) \mid x_i \geq z + 1\}$, where $z \in \mathbb{N}$. Choosing

**Algorithm 2** Benson's outer approximation algorithm

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

the variable and the bound, $z$, is a non-trivial task, as the performance of the MOBB highly depends on these choices. In the single-objective case, the lower bound (set) usually consists of a single solution and as a result, each variable $x_i$ takes a single value. This can be used to make an easy choice regarding the bound imposed (e.g. $z = \lfloor x_i \rfloor$). In the multi-objective case, multiple points may exist in the lower bound set, and as a consequence, a variable may take different values for different points. A trivial choice does not exist anymore, and a rule should be applied (see Section 5).

# 4 Linear relaxation for MOBB

In this section, we provide a strategy for accelerating the computation of the lower bound set (line 4 in Algorithm 1), i.e. the linear relaxation. Our methodology relies on Benson's outer approximation algorithm (Benson, 1998) and its recent refinements (see e.g. Csirmaz, 2015; Hamel, Löhne, and Rudloff, 2013; Löhne and Weißing, 2020). For this purpose, we need a formal definition of the concept of *outer approximation*.

**Definition 3.** *Let $\mathcal{P}, \mathcal{Q} \subset \mathbb{R}^p$ be two polyhedra such that $\mathcal{Q}_N \subseteq \mathcal{P}$. Then $\mathcal{P}$ is an outer approximation of $\mathcal{Q}$.*

An outline of a Benson-like algorithm is given in Algorithm 2. The algorithm works by iteratively building tighter outer approximations of $\mathcal{P}_{\leqq}^{LP} = \mathcal{Y}^{LP} + \mathbb{R}^p_{\leqq} = \mathcal{Y}_N^{LP} + \mathbb{R}^p_{\leqq}$ and starts with an initial polyhedron that contains $\mathcal{P}_{\leqq}^{LP}$. Next, half-spaces are iteratively found whose corresponding hyperplanes define facets of $\mathcal{P}_{\leqq}^{LP}$ until all the facets have been enumerated. The algorithm provides both a vertex-ray representation and a half-space representation of $\mathcal{P}_{\leqq}^{LP}$, where a pre-image is known for each of the vertices in $\mathcal{P}_{\leqq}^{LP}$.

The initialization step (line 1 of Algorithm 2) consists of finding an initial polyhedron that contains $\mathcal{P}_{\leqq}^{LP}$. At each iteration of Algorithm 2, if there exists a vertex $v$ in the vertex-ray representation $\mathcal{P}_V$ which is not included in $\mathcal{P}_{\leqq}^{LP}$, a cutting plane should be computed in order to separate $v$ from the polyhedron. In order to check the

**Algorithm 3** Updating the outer approximation (`updateP`)

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

inclusion of vertex $v$ on line 2, the linear program $F(v)$ is solved:

$$\begin{aligned} &\min s \\ &\text{s.t. } Ax \geqq b, && (1)\\ &\quad Cx - s \leqq v, && (2)\\ &\quad x, s \geqq 0 \end{aligned}$$

If the optimal value is 0, then $v \in \mathcal{P}_{\geqq}^{LP}$ and a pre-image of $v$ is obtained by storing the optimal values of the $x$ variables of $F(v)$; otherwise, $v$ is not included in $\mathcal{P}_{\geqq}^{LP}$. Let $u \in \mathbb{R}^m$ be optimal dual values corresponding to (1) and $w \in \mathbb{R}^p$ dual values corresponding to (2). Hamel et al. (2013) showed that the hyperplane $\mathcal{H} = \{y \in \mathbb{R}^p \mid w^T y = b^T u\}$ defines a facet of $\mathcal{P}_{\geqq}^{LP}$ and that $\mathcal{H}$ separates $v$ from $\mathcal{P}_{\geqq}^{LP}$. Hence, the hyperplane $\mathcal{H}$ on line 3 can be found using the dual values of $F(v)$. Once a cutting plane $\mathcal{H}$ is computed, the outer approximation of $\mathcal{P}_{\geqq}^{LP}$ is updated using function `updateP` on line 4 of Algorithm 2. The loop is repeated until no vertex $v$ can be found, and the algorithm stops ($\mathcal{P}_{\geqq}^{LP}$ has been found, line 6).

A description of `updateP` is given in Algorithm 3. As input, the algorithm takes the current half-space and vertex-ray representation and the cutting hyperplane. First, the vertex-ray representation is updated by examining adjacent vertices, finding new vertices of the facet of the hyperplane $\hat{\mathcal{H}}$ and removing old vertices not part of the polyhedron (line 3). Updating the vertex-ray representation using function `updateV` is known as

a sub-procedure of an *online vertex enumeration problem*. A well-known technique for solving this problem is the double description method (see e.g. Fukuda and Prodon (1996)).

Next, redundant faces are removed on lines 4-12. If redundant half-spaces are not removed, many unnecessary operations will be performed, e.g. when performing dominance tests. Moreover, having no redundant half-spaces is a necessary condition for finding adjacent vertices in the vertex enumeration algorithm used (Fukuda and Prodon (1996)). Since $\mathcal{P}^{LP}_{\leqq}$ is a full-dimension polyhedron, facets are of dimension $p-1$, and all faces with dimensions below $p-1$ are redundant. Consequently, if a facet is defined by $p-1$ vertices and rays or less, then it is redundant. This is checked on lines 5-6. Even though this is a necessary condition for any $p$, it is not a sufficient condition when $p > 3$. Indeed, in this case, a face of dimension 2 can be defined by more than $p-1$ vertices. A face of dimension $d$ can be described as the intersection of at least $p-d$ hyperplanes (Nemhauser and Wolsey, 1999). Hence a face of dimension $d < p-1$ is the intersection of two or more hyperplanes. Since the input $\mathcal{P}_H$ to Algorithm 3 only contains facets, the only way for a facet to become a face is if it gets intersected with the new cutting hyperplane $\hat{\mathcal{H}}$ such that all of its vertices and rays are located on $\hat{\mathcal{H}}$ (lines 8-9).

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

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

**Lemma 2.** *Consider Algorithm 2 and let $\mathcal{P}^0$ denote the initial polyhedron with half-space representation $\mathcal{P}^0_H$ (line 1). Then only half-spaces in $\mathcal{P}^0_H$ may be redundant for $\mathcal{P}^{LP}_{\leqq}$. Moreover, if $\mathcal{P}^0 = \{y^I_{LP}\} + \mathbb{R}^p_{\leqq}$ then all half-spaces in $\mathcal{P}^0_H$ are facets of $\mathcal{P}^{LP}_{\leqq}$.*

*Proof.* Hamel et al. (2013) showed that the cutting hyperplane $\mathcal{H}$ found on line 3 defines a facet of $\mathcal{P}^{LP}_{\leqq}$. Hence, only half-spaces in $\mathcal{P}^0_H$ may be redundant. Let $y^I_{LP} = (\hat{y}_1, \ldots, \hat{y}_p)$. If $\mathcal{P}^0 = \{y^I_{LP}\} + \mathbb{R}^p_{\leqq}$ then the half-spaces $\{y \in \mathbb{R}^p \mid y_i \geq \hat{y}_i\}, i = 1, \ldots, p$ define the facets of $\mathcal{P}^0$, which are facets of $\mathcal{P}^{LP}_{\leqq}$ too. □

## 4.1 Warm-starting Benson-like algorithms in MOBB

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

**Lemma 3.** *Consider a child node $\eta^c$ of the father node $\eta^f$ in the branch-and-bound tree of Algorithm 1. Then $\mathcal{P}^{LP}_{\leqq}(\eta^f) := \mathcal{Y}^{LP}_N(\eta^f) + \mathbb{R}^p_{\leqq}$ is an outer approximation of $\mathcal{P}^{LP}_{\leqq}(\eta^c) := \mathcal{Y}^{LP}_N(\eta^c) + \mathbb{R}^p_{\leqq}$.*

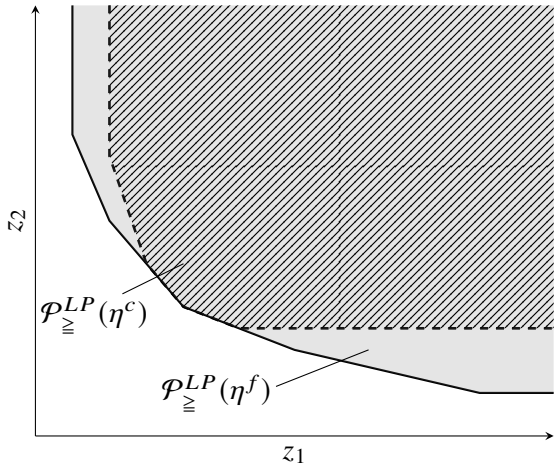


Figure 1: The lower bound set of the father node $\eta^f$ is an outer approximation of the lower bound set of the child node $\eta^c$.

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

Due to Lemma 3, polyhedron $\mathcal{P}^{LP}_{\geqq}(\eta^f)$ can be used as the initial outer approximation when starting Algorithm 2 in a child node $\eta^c$ (see Figure 1). That is, at any child node, it is possible to *warm-start* the computation of the linear relaxation by using the relaxation found in the father node. As a result, the total number of linear programs to be solved is expected to decrease since the only way to obtain a facet is to solve $F(v)$ for a vertex $v$ and obtain an optimal value strictly larger than zero. Hence a facet that is present both in $\mathcal{P}^{LP}_{\geqq}(\eta^f)$ and $\mathcal{P}^{LP}_{\geqq}(\eta^c)$ will be enumerated only once, since it is already known when starting Algorithm 2 in child node $\eta^c$. However, some half-spaces in $\mathcal{P}_H$ may have to be removed in Algorithm 2 since they define non-facet faces and are therefore redundant. Due to Lemma 2 we have:

**Corollary 1.** *Consider Algorithm 1 using Algorithm 2 to find the lower bound set on line 4. If we use initial outer approximation $\mathcal{P} = \{y^I_{LP}(\eta^c)\} + \mathbb{R}^p_{\geqq}$ at the root node $\eta^0$, then no redundant half-spaces have to be removed from $\mathcal{P}_H$ during Algorithm 2. If we use initial outer approximation $\mathcal{P} = \mathcal{P}^{LP}_{\geqq}(\eta^f)$ at a child node $\eta^c$ with father node $\eta^f$, then only half-spaces of $\mathcal{P}^{LP}_{\geqq}(\eta^f)$ may be redundant.*

Due to Corollary 1 we initialize Algorithm 2 with outer approximation $\{y^I_{LP}(\eta^c)\} + \mathbb{R}^p_{\geqq}$ in the root node and hence do not have to check for redundant half-spaces (lines 4-12 in Algorithm 3). Moreover, when using $\mathcal{P}^{LP}_{\geqq}(\eta^f)$ as initial outer approximation in a child node, only the half-spaces of $\mathcal{P}^{LP}_{\geqq}(\eta^f)$ have to be checked for redundancy.

**Additional advantages of warm-starting the lower bound computations**

Warm-starting Algorithm 2 when solving the linear relaxation in a MOBB brings additional information that can be utilized to accommodate additional speed-ups by eliminating redundant work. In the following, we present a number of observations that can help to speed-up the processing of branching nodes. In what follows, we will assume that branching is performed by adding simple bounds to variables, i.e. $x_i \leq z$ or $x_i \geq z$, for $z \in \mathbb{N}_0^n$. Further, we will refer to $\eta^c$ as a child node of the father node $\eta^f$.

In Algorithm 2, to confirm that a vertex $v$ is feasible, the linear program $F(v)$ has to be solved and have an optimal value of 0. However, this is not always necessary when using the polyhedron $\mathcal{P}_{\leqq}^{LP}(\eta^f)$ as a starting outer approximation when computing $\mathcal{P}_{\leqq}^{LP}(\eta^c)$:

**Observation 1.** *After solving the linear relaxation in $\eta^f$, a pre-image for each vertex of $\mathcal{P}_{\leqq}^{LP}(\eta^f)$ is known. Hence, when solving the linear relaxation $P^{LP}(\eta^c)$ using $\mathcal{P}_{\leqq}^{LP}(\eta^f)$ as an initial outer approximation, it is known that $F(v) = 0$ for all $v \in \mathcal{P}_{\leqq}^{LP}(\eta^f)$ with a feasible pre-image in $\eta^c$.*

This significantly reduces the number of single-objective linear programs that needs to be solved in Algorithm 2. In addition, when branching as stated in Section 3, we know that from $\eta^f$ to $\eta^c$, only one constraint in the form $x_i \leq z$ or $x_i \geq z+1$ is added, $z \in \mathbb{N}$. Hence, verifying the feasibility of a pre-image of a vertex reduces to simply comparing the value of a variable to a constant. Note that if more involved branching constraints are used, it is necessary to check that the pre-image satisfies all of them.

Observation 1 only holds in $\eta^c$ for the vertices that belong to $\mathcal{P}_{\leqq}^{LP}(\eta^f)$. A vertex $v$, generated *during* the execution of Algorithm 2 for $\eta^c$, has to be checked for feasibility by solving $F(v)$, because we do not know a pre-image for it yet. Furthermore, we still need to solve $F(v)$ for any vertex $v \in (\mathcal{P}_{\leqq}^{LP}(\eta^f))_V$ that does not have a feasible pre-image for $\eta^c$ in order to generate the cutting plane that cuts off $v$ in line 3 of Algorithm 2.

Regarding the update of the upper bound set, here the incumbent set, there is also an easily achievable speed-up available: for any integer feasible vertex $v \in ((\mathcal{P}_{\leqq}^{LP}(\eta^c))_V \cap (\mathcal{P}_{\leqq}^{LP}(\eta^f))_V$ it has already been checked whether it improves the current upper bound set or not when processing $\eta^f$. Thus, only newly generated pre-images of vertices should be used when updating the upper bound set in line 5 of Algorithm 1 at node $\eta^c$.

Similarly, by keeping track of the cutting planes generated in node $\eta^c$, one can reduce the number of comparisons done when performing the dominance test in line 9 of Algorithm 1. Let $u \in \mathcal{N}(\mathcal{U})$ be a local upper bound dominated by the lower bound set in the father node $\eta^f$ of $\eta^c$. This implies that $u \in \mathcal{P}_{\leqq}^{LP}(\eta^f)$, which, by using the definition of a polyhedron in terms of half-spaces, is equivalent to $u \in \bigcap_{h \in (\mathcal{P}_{\leqq}^{LP}(\eta^f))_H} h^+$. Hence, we already know that in $\eta^c$, for all the old supporting hyperplanes $h \in (\mathcal{Y}_N^{LP}(\eta^c) + \mathbb{R}_{\leqq}^p)_H \cap (\mathcal{P}_{\leqq}^{LP}(\eta^f))_H$, we have $u \in h^+$. As a result, the only way for $u$ to become non-dominated is to be located outside the half-space corresponding to one of the new facets generated in $\eta^c$. Otherwise, it remains dominated. Hence, the status of

$u$ can be determined by looking at the facets generated in $\eta$ only.

To conclude, it is possible to derive additional information using Algorithm 2 because its starting point is exactly its ending point in its father node. Hence, we can easily keep track of how the lower bound set was modified by the new constraints generated from the father to the child node, and use this information to reduce the number of redundant operations.

# 5 Computational experiments

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

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

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

## 5.1 Implementation details and algorithm configurations

All algorithms are implemented in C++17 and compiled using the MSVC compiler with default options. Experiments are carried out on a personal computer with an Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz processor and 8GB of RAM memory, using Windows 10. The algorithms are available at `https://github.com/NicolasJForget/LinearRelaxationBasedMultiObjectiveBranchAndBound/tree/v1.0`. Different configurations of Algorithm 1 will be tested:

**Node selection:** A breadth-first strategy is used on line 3 of Algorithm 1. Preliminary experiments showed that there is no clear winner between using a depth-first or a breadth-first strategy. On the set of instances we

considered, breadth-first performed slightly better on average, and we will stick to that choice for the rest of the experiments.

**Calculation of lower bound set:** The lower bound set on line 4 of Algorithm 1 is computed using two different configurations:

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

Calls to the single-objective linear programming solver in Algorithm 2 are performed with CPLEX 12.10, using and modifying a single model for the whole branching tree. When the model is modified, the solution process is initialized using the optimal basis of the previous model solved. The lower bound polyhedron is stored using a data structure with a linked vertex-ray and a half-space representation, and updated using Algorithm 3.

**Updating the upper bound set:** The upper bound set is updated by searching for vertices in the lower bound set of a node with an integer-feasible pre-image (line 5 of Algorithm 1). Each time a new point is added to the upper bound set, the set of local upper bounds is also updated using the algorithm developed by Klamroth et al. (2015).

**Pruning nodes:** A node is checked for pruning (line 6 of Algorithm 1) by first checking if the node can be pruned by infeasibility, then by optimality, and finally by dominance using Lemma 1. The dominance test terminates when one dominated local upper bound is found, or when all local upper bounds have been checked. Moreover, a non-dominated local upper bound in the father node will remain non-dominated in all of its child nodes. Hence, it is not necessary to check it again. However, this requires to keep track locally of the status of each local upper bound in $\mathcal{N}(\mathcal{U})$, which evolves globally. Preliminary experiments showed that if that resulted in minor improvements when $p = 3$ (a reduction of a few percentage points of total CPU time), the computational cost was greater than recomputing the dominance test from scratch in each node for larger $p$, due to the larger number of local upper bounds. Consequently, no information is kept from the father node to the children node, and the dominance test is performed from scratch at each node.

**Variable selection:** The variable chosen for branching on line 7 of Algorithm 1 is the variable that is the most often fractional among the vertex solutions in the lower bound set. If no fractional variable exists, we select the free binary variable with the average value closest to 0.5. If there is no free binary variable,

a general integer variable with different maximum and minimum values is chosen. If there are ties, the variable with the lowest index is chosen. Given branching variable $x_i$, two child nodes are created using bound $z$. If $x_i$ is binary, bound $z = 0$ is used, i.e. a rule denoted `BINARY` that branches on $x_i = z$ and $x_i = z + 1$. Given an integer branching variable $x_i$, a bound $z \in \mathbb{N}$ has to be chosen, and we branch using constraints $x_i \leq z$ and $x_i \geq z + 1$. Let $\{x_i^1, \ldots, x_i^k\}$ denote the sorted $k$ values in the pre-images of all the vertices of the lower bound set. We test the configurations:

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

In principle, using `LP` vs `WLP` should not affect the branching tree given all other configurations fixed. However, when the pre-image of a vertex is in fact feasible, the `LP`-configuration might produce an alternative optimum when solving $F(v)$ whereby a different pre-image of the vertex $v$ is found. This will, potentially, lead to faster updates of the upper bound set and different search paths being followed as the pre-images are used to decide on the branching variable as well as on the bound. Our test showed, however, that these differences only affect very few instances, and that the effect is negligible. As a result we only test the most promising rule (`MOFV`) for finding the bound (variable selection) for `LP`.

## 5.2 Test instances

Different problem classes are considered with 3, 4, and 5 objective functions. An overview is given in Table 1. For each problem class, size (number of variables), and number of objectives, 10 instances are generated. All instances can be obtained from `https://github.com/MCDMSociety/MOrepo-Forget21` and `https://github.com/MCDMSociety/MOrepo-Kirlik14`.

The problem class, denoted by ILP, consists of randomly generated MOILPs with up to 40 variables. These instances were proposed and solved in Kirlik and Sayın (2014). Note that in general, the constraint matrix for integer models modeling real-life applications is sparse and structured. We use these instances to investigate how the algorithm performs on dense and unstructured integer models. Class PPP consists of *Production*

Table 1: Instances used (480 instances in total).

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

[a] Number of objectives.

[b] Variable sizes.

[c] Range of the objective function coefficients $C$.

[d] Percentage of objective coefficients not dominated by other coefficients.

[e] Percentage of non-zeros in the constraint matrix $A$.

[f] Number of instances.

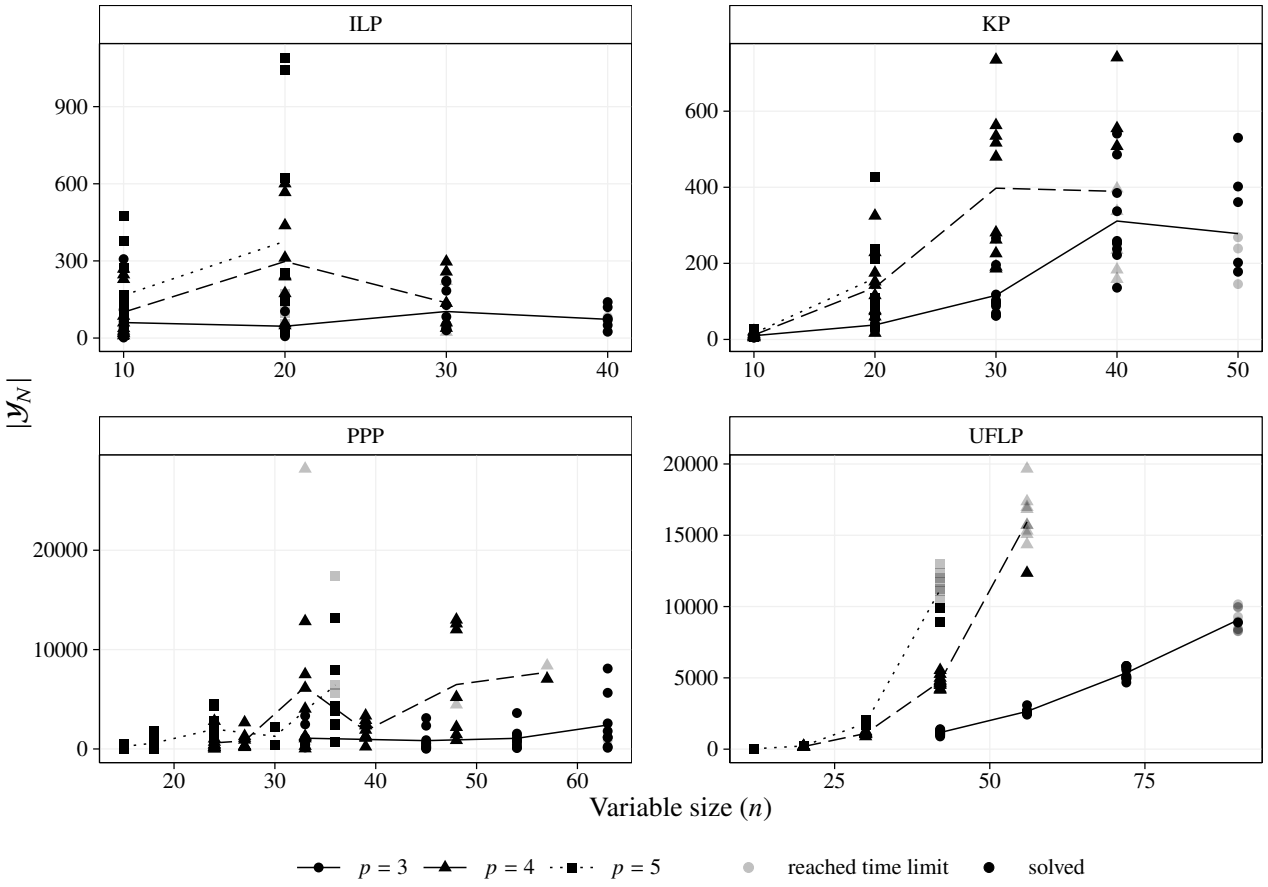


Figure 2: Number of non-dominated points. One point for each instance and average lines are given. Instances that have not been solved to optimality are illustrated with a transparent point.

*Planning Problems* with up to 63 variables. Both integer and binary variables are included in the model, as well as "big $M$" constraints. We refer the reader to Appendix B.1 for a model description and more details regarding the ranges of the coefficients. Class KP are binary *Knapsack problems* with up to 50 variables/items. These instances were proposed and solved in Kirlik and Sayın (2014). Finally, class UFLP are *Uncapacitated Facility Location Problems* with up to 90 variables. It is a combinatorial problem (only binary variables), and the objective coefficients were generated such that the percentage of objective coefficients not dominated by other coefficients is high (approx. 80%). This results in many non-dominated points. We refer the reader to Appendix B.2 for more details regarding the model and the generation of coefficients. Note that PPP and UFLP have a sparse constraint matrix compared to ILP.

In Figure 2 the number of non-dominated points are given for each instance. We have increased the variable size for each problem class until the size becomes so large that some instances cannot be solved within the time limit. The instances which have not been solved to optimality (19%) are illustrated with transparent points. In general the number of non-dominated points grows with variable size ($n$) and number of objectives ($p$). Note though that there may be a high variation for fixed $n$ and $p$. Moreover, the variation grows with $n$ and $p$. The number of non-dominated points seems to be correlated with the density of the constraint matrix (%A in Table 1). A dense constraint matrix may in some cases result in a small feasible solution space corresponding to few non-dominated points. However, more important factors is the range of the objective coefficients and the percentage of non-dominated objective coefficients (consider PPP and UFLP). Given a problem class, increasing these factors will result in a higher number of non-dominated points (Forget, Nielsen, and Gadegaard, 2020). Moreover, for UFLP the number of non-dominated points grows rapidly as a function of the number of variables, which is due to the high percentage of objective coefficients not dominated by other coefficients.

### 5.3 Performance of the different algorithm configurations

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

A plot of the CPU time for each instance is given in Figure 3. Note the variation in CPU time for the 10 instances given each class and variable size. Warm-starting the computation (`WLP`) in general performs better than `LP`. On average `WLP` (using the best variable selection configuration) performed 29% faster compared to `LP`. This can also be seen in Figure 4 illustrating the number of solved instances given a CPU time limit. We have increased the variable size for each problem class until the size becomes so large that some instances cannot be

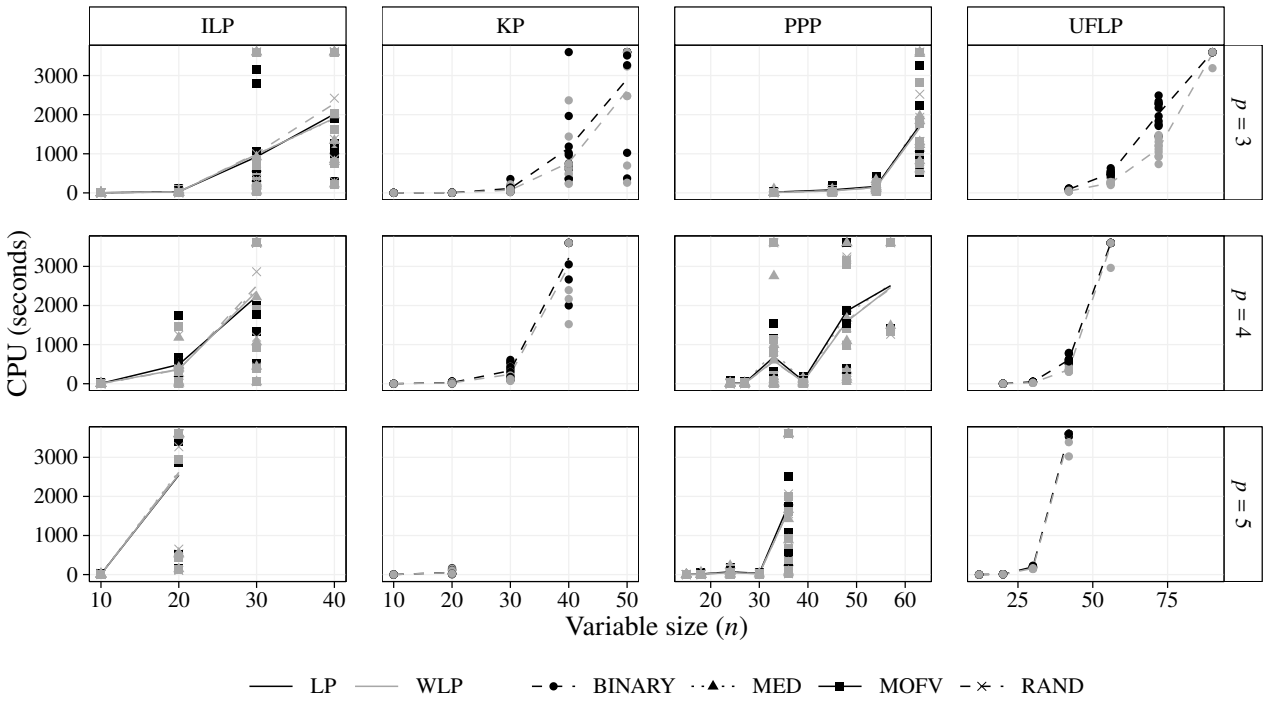


Figure 3: CPU times (a point for each instance) with average lines.

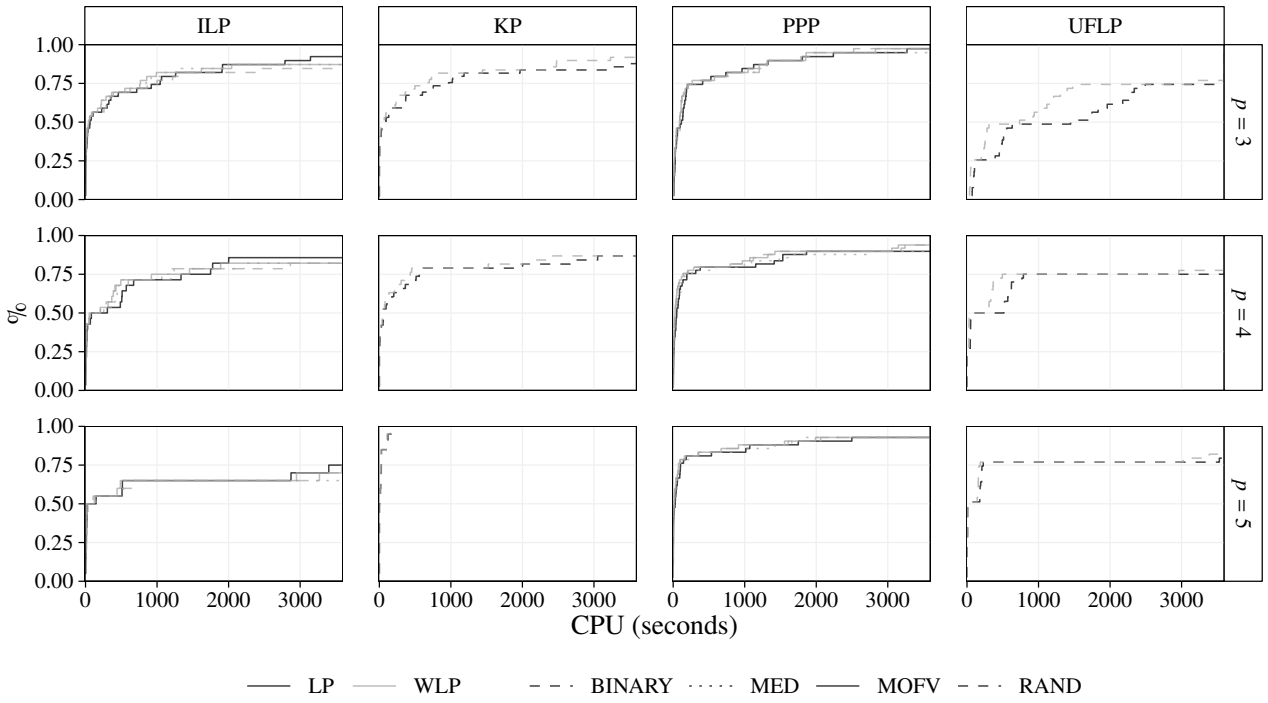


Figure 4: Number of instances in percent solved given cpu time (seconds). An instance is considered as unsolved if the cpu time exceed 3600 seconds (time limit). One curve is depicted for each of the configuration tested.

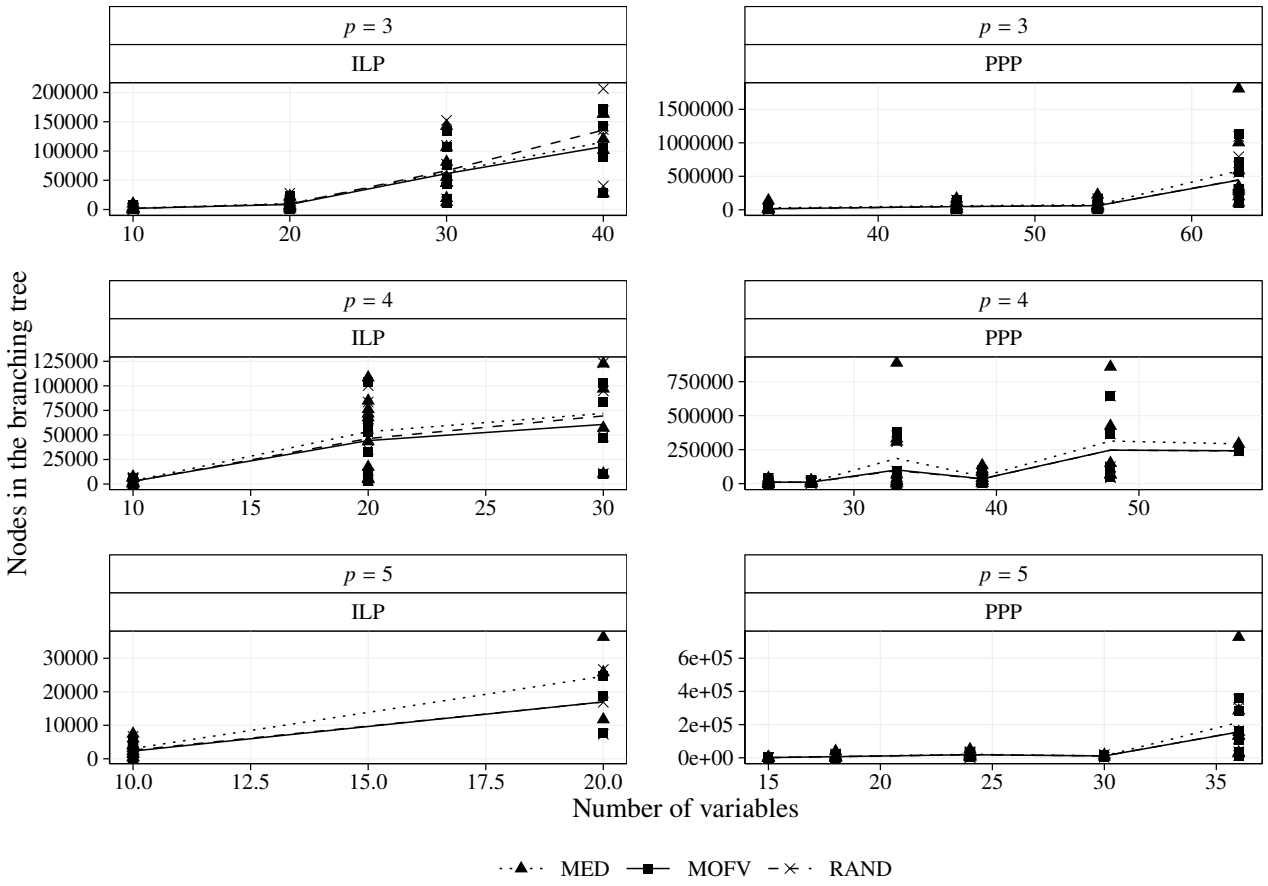


Figure 5: Tree size for different variable selection rules with average lines.

solved within the time limit. That is, the number of instances solved before the time limit is below 100%.

In a few instances `LP` performed best (fastest in 4% of the instances). We will have a closer look at the reason for this in Section 5.5

## 5.4 Variable selection - Rules for choosing the bound

We are here interested in determining whether one rule for finding the bound (`MED`, `RAND`, and `MOFV`) is consistently better than the other when considering non-binary integer problems. Since the effect on the branching tree of using `LP` vs `WLP` is negligible, we will consider the `WLP` configuration here. By considering the performance profiles in Figure 4 we see that there is no clear winner among `MED`, `RAND`, and `MOFV`.

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

If we take a look at the size of the branching tree in Figure 5, then the tree size for `MOFV` is not bigger than the one for `MED`. As a result, we use `MOFV` in the succeeding experiments, since `MOFV` is slightly faster on average and produces the smallest branching tree.

Table 2: Speed-up factor using WLP instead of LP for each problem class and number of objectives. The rule for choosing the bound is MOFV.

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

### 5.5 Detailed performance of different algorithm parts

In this section, we take a closer look at different parts of Algorithm 1. Different speed-up factors by using WLP instead of LP are given in Table 2. The factor is obtained by dividing the LP value with the WLP value. Only instances with both configurations solved are recorded. WLP are on average 1.47 times faster than LP with significant differences among the problem classes, e.g. for problem class UFLP, WLP is on average 1.61 times faster while for class ILP the speed-up is 1.35.

Most of the CPU time (95% for LP and 91% for WLP) is used on calculating the lower bound set (Algorithm 2) and the speed-up is mainly due to a reduction in the number of times the linear programming solver has to be called on line 2 in Algorithm 2. This can be seen in Table 2. For example, for UFLP WLP is 1.61 times faster and solves 2.11 times less linear programs on average than LP. However, when using WLP the initial outer approximation has to be copied from the father node into the child node and managing the polyhedron is harder since we have to check for redundant half-spaces in Algorithm 3 (lines 4-12). As a result we have a smaller reduction in CPU times than the reduction in number of LPs solved.

Since most of the time is used for calculating the lower bound set, let us have a closer look at the relative usage of the different parts of Algorithm 2. An overview is given in Figure 6 where the different parts are:

**Initialization** Proportion of time used to calculate the outer approximation with polyhedron $(\{y^I_{LP}\} + \mathbb{R}^p_{\geqq}$ for LP, and time to copy the lower bound set from the father node for WLP. That is, time used to find the input on line 1 of Algorithm 2.

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

**Update polyhedron** Proportion of time used for updating the polyhedron using Algorithm 3.

**Other** Proportion of time used on other parts of Figure 6, such as picking a vertex in the polyhedron (line 2), retrieving pre-images from CPLEX's ouput, checking the pre-image of a point from the father node...

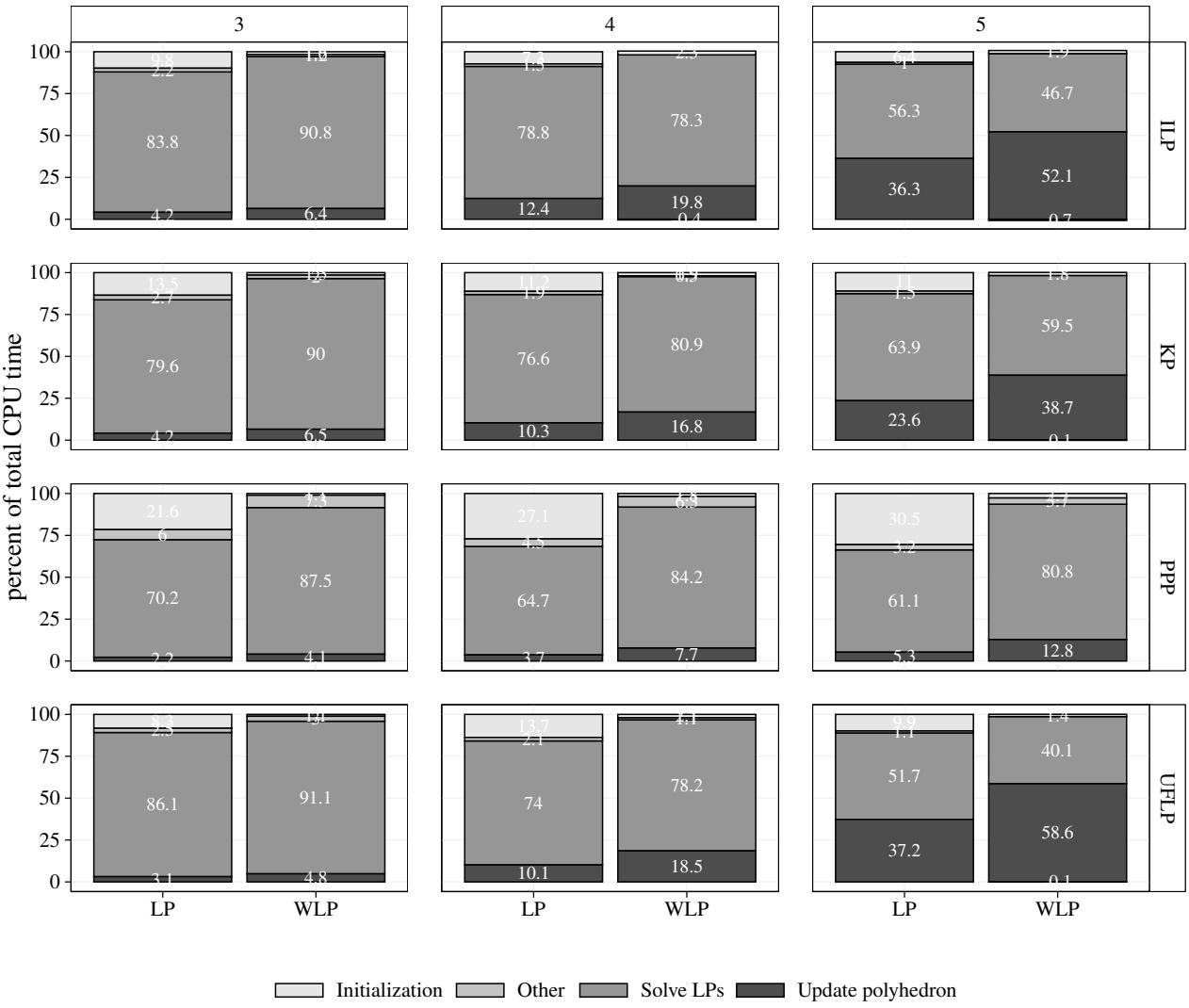


Figure 6: Proportion of the cpu time spent in the different components of Algorithm 2.

First, note that the proportion of time for initialization is much higher for `LP` compared to `WLP`. For `WLP` the time for copying the lower bound set from the father is negligible. Second, note that the cost of updating the polyhedron increases with the number of objectives. This is an observation that was also made by Csirmaz (2015), who showed that in higher dimensions ($p = 10$ in their paper), updating the polyhedron was actually the bottleneck of the Benson-like algorithm. This also explains why the speed-up factors in Algorithm 1 in general decrease with the number of objectives while the reduction of linear programs solved in general increase. Even though we solve relatively less linear programs for increasing $p$, we have to use more time on updating the polyhedron containing the lower bound set. Hence alternative lower bound sets that does not require to manage a polyhedron, or at least less polyhedral operations, may be preferred in higher dimensions. Next, note that updating the polyhedron takes a higher proportion of time for `WLP`. This is because we have to check for

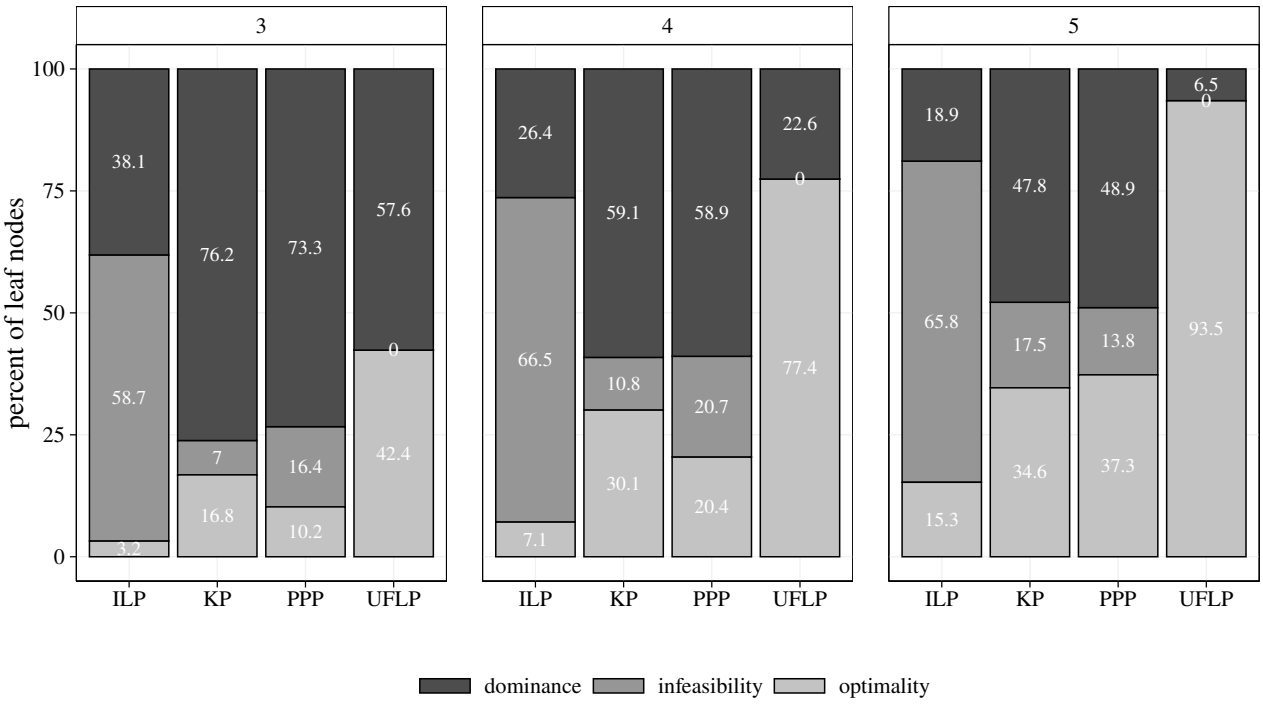


Figure 7: Proportion of leaf nodes pruned by infeasibility, optimality and dominance.

redundant half-spaces (lines 4-12 in Algorithm 3). Finally, observe that even though we have a high reduction in the number of linear programs solved for `WLP` the proportion of time used for solving linear programs is still the most predominant (except for UFLP, $p = 5$).

## 5.6 Pruning nodes

In this section we take a closer look at how nodes are pruned. Recall that a node is checked for pruning by first checking if the node can be pruned by infeasibility, next optimality, and finally by dominance. The results are illustrated in Figure 7 where the proportion of leaf nodes pruned by infeasibility, optimality and dominance are given.

It appears from Figure 7 that different behaviors are observed for the different problem classes. However, for all problem classes, the proportion of leaf nodes pruned by dominance decreases as the number of objectives increases. The reason for this is that the likelihood of a point is non-dominated increases as we add dimensions to the objective space, which makes the nodes harder to prune by dominance.

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

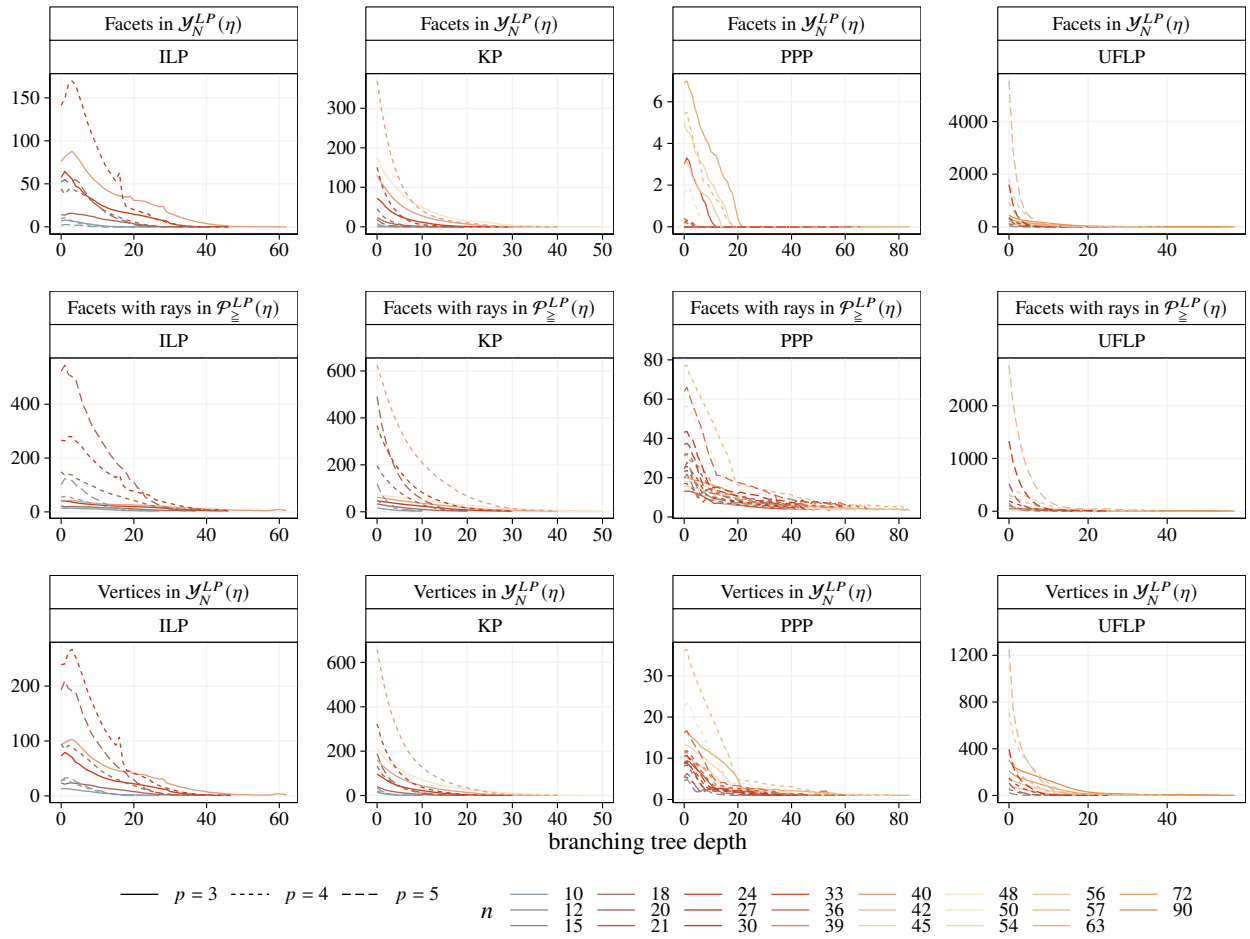


Figure 8: Average number of facets and vertices in the lower bound set and polyhedron calculated using Algorithm 2.

the number of objectives increases, generating all non-dominated outcomes comes closer to a total enumeration of the decision space.

## 5.7 Geometric properties of the lower bound set during the algorithm

In this section, we take a closer look at the polyhedral properties of the lower bound sets found at each node in the branching tree during the algorithm. In Figure 8 statistics about the number of facets and vertices in $\mathcal{Y}_N^{LP}(\eta)$ and facets with rays in $\mathcal{P}_{\geqq}^{LP}(\eta)$, i.e. extra facets needed for the full-dimensional polyhedron $\mathcal{P}_{\geqq}^{LP}(\eta)$ is given. The numbers are given as a function of the depth of the branching tree with a line for each $p$ and $n$.

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

Second, consider a problem class and a fixed number of variables $n$. As the number of objectives grows, the lower bound sets contain more facets and vertices. That is, more objectives generate more complex lower bound

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

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

Next, compare the number of vertices among problem classes. The number of vertices for PPP is lowest and highest for UFLP. That is, the lower bound sets in PPP are relatively simple compared to the UFLP and hence much faster to calculate and update (see Figure 6). Indeed, the proportion of cpu time spent in finding the initial polyhedron is very large compared to other problem classes, which suggests that only a few iterations are required to solve the linear relaxation once the initial polyhedron is found. That is particularly beneficial to `WLP` since, as we observe in Figure 6, warm-starting Algorithm 2 significantly reduces the initialization part of the algorithm.

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

Finally, recall the size of the non-dominated sets in Figure 2. Here the size of the non-dominated set for PPP is high. However, the number of vertices in the lower bound set is low. That is, problems with large non-dominated sets may generate (often weak) lower bound sets with few vertices. The opposite is also true. For ILP and KP the number of vertices is relatively higher compared to the size of the non-dominated set. That is, the relationship between the number of vertices and number of non-dominated points (i.e. the upper bound set) is problem specific.

### 5.8 Proving optimality

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

In order to investigate the potential of such a strong upper bound set found early, we have run the algorithm

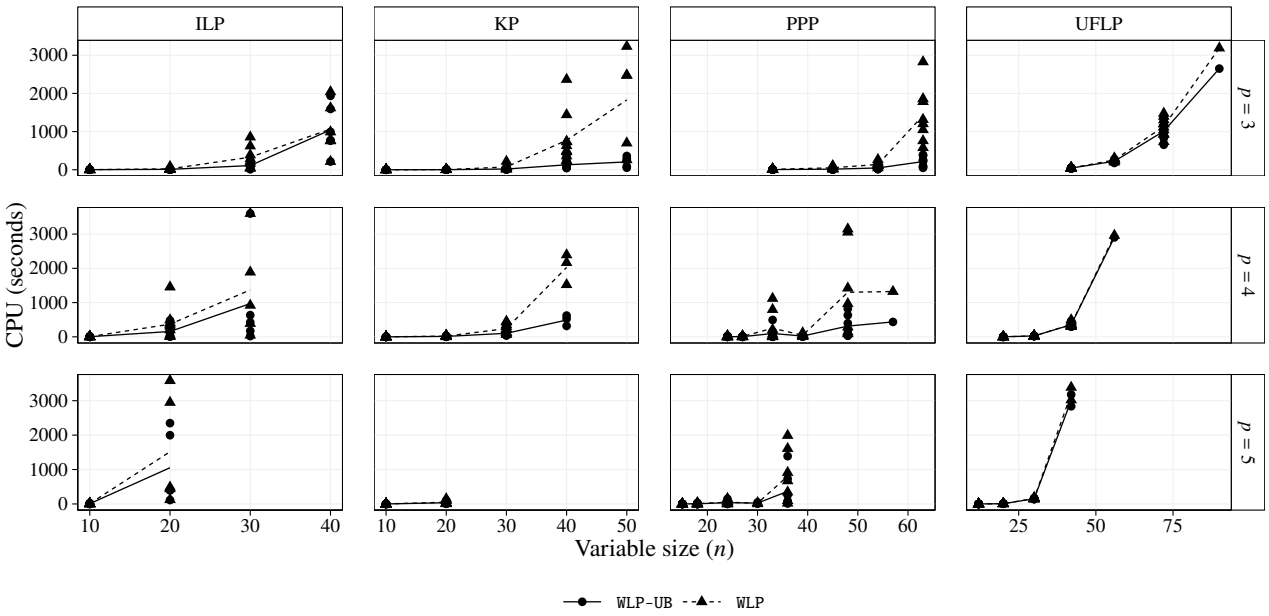


Figure 9: CPU times (with average lines) for WLP and WLP with the non-dominated set used as initial upper bound set in the root node (WLP-UB).

where the upper bound set is initialized by $\mathcal{Y}_N$ (using WLP with MOFV or BINARY). That is, on line 1 of Algorithm 1 we replace $\mathcal{U} \leftarrow \emptyset$ with $\mathcal{U} \leftarrow \mathcal{Y}_N$. This corresponds to having a very fast and effective heuristic.

As can be seen from Figure 9 the reduction in CPU time is significant. On average over all the instances the speed-up factor is 2.06 meaning that on average, not providing an optimal solution at the root node, makes the cpu time increase with 106%.

As there is no significant computation time involved in generating solutions in our branch-and-bound algorithm (feasible solutions are simply harvested from integer feasible vertices of the lower bound sets), the speed-up must come from the increased pruning potential. The number of nodes in the branching tree on average decrease with 30% for WLP-UB. Note also, that in the WLP-UB configuration, the algorithm still check whether each integer feasible vertex of the lower bound sets found should enter the upper bound set, which underlines the fact that the reduction in computation time comes from the increased pruning potential.

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

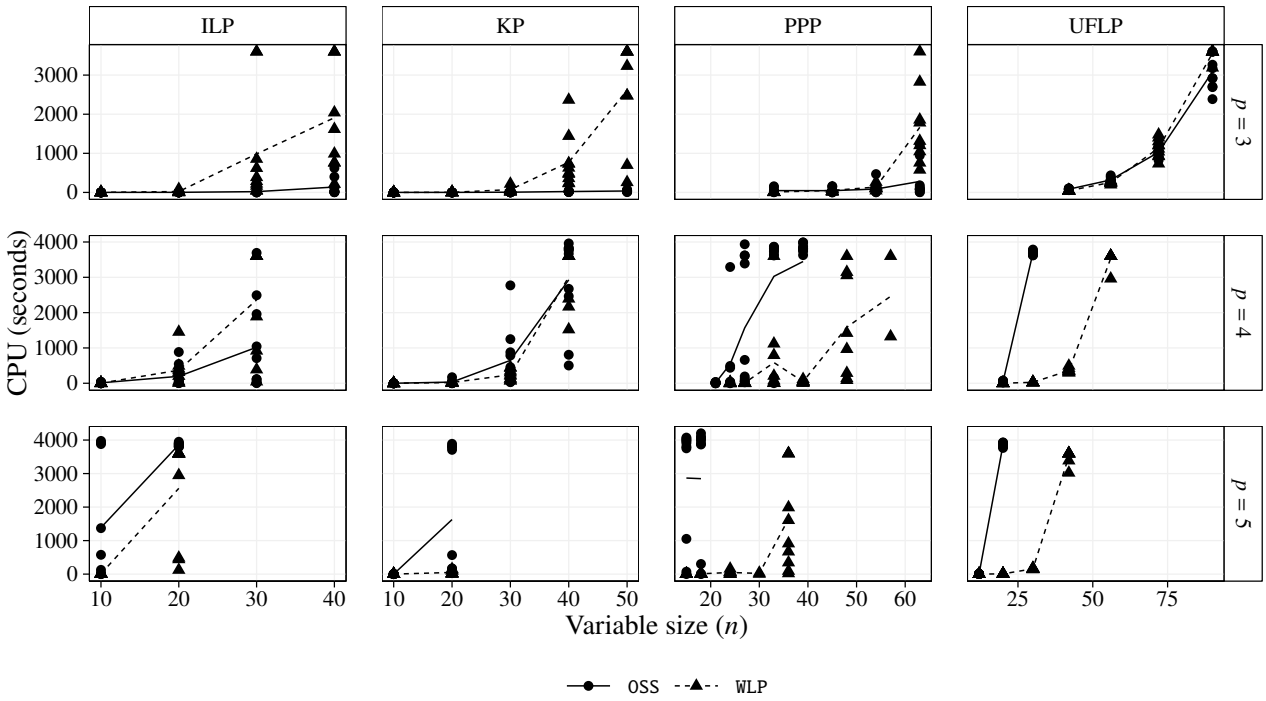


Figure 10: CPU times for `OSS` and `WLP` (using `MOFV`) with average lines.

## 5.9 Performance of the multi-objective B&B algorithm compared to an objective space search algorithm

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

The current development of OSS algorithms is more mature and the algorithms may outperform multi-objective B&B algorithms. Moreover, OSS algorithms benefit 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 against an OSS algorithm. Indeed, we do believe that it will take further research before reaching a competitive B&B algorithm, and we propose here a first step towards an efficient implementation. Moreover, our research may result in an advancement of the development of promising methods that hybridize decision space and objective space search methods.

The code by Kirlik and Sayın (2014) was updated to work with CPLEX 12.10 and compiled using the MSVC compiler with default options. The results are shown in Figure 10. The OSS algorithm is tested against

the best configuration of our branch-and-bound framework (WLP with MOFV or BINARY). Different winners with respect to CPU time are observed. First, it appears that this OSS algorithm performs better on instances with three objective functions. However, as the number of objective functions increase, the trend is reversing and the B&B algorithm becomes better than the OSS algorithm for all problem classes when $p = 5$.

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

To test the algorithms with longer running time, we extended the time limit to 4 hours, and considered two unsolved instances (either for WLP or OSS) for each problem class and number of objectives. The two instances with the lowest and highest upper bound set was picked among the unsolved instances. If only one instance was unsolved, we chose the instance solved with the largest cpu time for WLP for the second instance. The results are given in Table 3 containing CPU times, speedup ratios and approximation measures.

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

Consider Table 3. First, observe even beyond the one hour time limit, the B&B algorithm exhibit the same behavior as when given the limit. The OSS algorithm performs better on instances with three objective functions and as the number of objective functions increase, the trend is reversing and the B&B algorithm becomes better than the OSS algorithm. Indeed, B&B is most competitive compared to the OSS algorithm on problems that exhibit a large number of non-dominated points.

Second, for UFLP, the proportion of non-dominated points obtained within one hour is very high even though the total computation time is not necessarily that close to one hour. This shows that for this problem class, the B&B algorithm spend a lot of time obtaining a few missing non-dominated points and trying to prove optimality. This observation confirms the results from Figure 9, where having $\mathcal{Y}_N$ as upper bound set did not significantly reduce the total CPU time for UFLP. Also, it implies that in some cases, stopping the B&B prematurely is likely to provide a good approximation of $\mathcal{Y}_N$. This is not necessarily the case if an OSS

Table 3: Some instances solved with a time limit increased to 4 hours. KP for $p = 5$ were not added since all instances are solved in less than one hour. ILP for $p = 5$ are not added since all instances resolved resulted in more than 4 hours CPU time for `WLP` and `OSS`.

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

[a] Number of objectives.

[b] Number of variables.

[c] Number of non-dominated points.

[d] CPU time when solving using `WLP`.

[e] CPU time when solving using `OSS`.

[f] Speedup ratio obtained by dividing the CPU time of `OSS` by the CPU time of `WLP`.

[g] Proportion of non-dominated points found after one hour of running `WLP`, in percentage.

[h] The upper bound set obtained after one hour is an $\epsilon$-approximation of $\mathcal{Y}_N$.The number reported is $100\epsilon$, and can be interpreted as a percentage of the distance between $y^I$ and $y^N$.

[i] The number reported is $100\bar{\epsilon}$, and can be interpreted as a percentage of the distance between $y^I$ and $y^N$.

[j] Size of the upper bound set at the 4 hours time limit.

algorithm is stopped early, since at most one non-dominated point is generated at each iteration whereas multiple solutions can be harvested at a single node in a B&B algorithm.

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

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

# 6 Conclusion

In Section 3, we implemented a branch-and-bound algorithm that can solve any MOILP with any number of objectives. It was inspired by the recent successful bi-objective frameworks found in the literature, and then adapted to the multi-objective case. In particular, it was based on the use of linear relaxations to generate lower bound sets, and used a Benson-like algorithm to do so. We also pointed out that in case integer variables exists in the problem solved, we need to chose the bound imposed on the branching variable in the child nodes in addition the variable to branch on. This decision is not trivial anymore when there exist two points or more in the lower bound set. We tested three different rules in Section 5 and showed that despite the fact that they performed quite similarly in practice, there are instances where the choice has a significant impact in terms of total CPU time.

Moreover, in Section 4 we proposed a way to accelerate the computation of the linear relaxation in the specific context of a multi-objective branch-and-bound algorithm. It relies on the use of the lower bound set from the father node to warm-start the solution process in the current node. Our experiments showed that this led to a great reduction in the number of calls to the single-objective linear programming solver, which resulted in a significant speed-up for most of our instances. However, warm-starting the lower bound set computation comes at a greater cost of managing the polyhedra coresponding to the lower bound sets. A consequence of that is a decrease in the speed-up as the number of objective functions increases.

This latter observation suggests that for high-dimensional problems, it may be preferred to use a lower bound set that does not require as many polyhedral operations as the one used in this paper. An alternative approach could be to use an implicit lower bound set as defined in Gadegaard et al. (2019) instead of explicitly computing the linear relaxation. In such an approach, line 4 of Algorithm 1 is skipped, and a linear program similar to $F(u)$ is solved to check whether a local upper bound $u$ is dominated by the lower bound set.

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

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

## References


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

P. Belotti, B. Soylu, and M. M. Wiecek. A branch-and-bound algorithm for biojbective mixed-intger programs. Technical report, Clemson University, 2013. URL `http://www.optimization-online.org/DB_HTML/2013/01/3719.html`.

P. Belotti, B. Soylu, and M.M. Wiecek. Fathoming rules for biobjective mixed integer linear programs: Review and extensions. *Discrete Optimization*, 22:341–363, nov 2016. doi:10.1016/j.disopt.2016.09.003.

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

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

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

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

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. Computational results (all instances). Technical report, Aarhus University, 2020. URL `https://mcdmsociety.github.io/MOrepo-Forget20/report.html`. Results for all the instances at the repository MOrepo-Forget20.

K. Fukuda and A. Prodon. Double description method revisited. In *Lecture Notes in Computer Science*, volume 1120, pages 91–111. Springer, Berlin, 1996.

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

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

N. Jozefowiez, G. Laporte, and F. Semet. A generic branch-and-cut algorithm for multiobjective optimization problems: Application to the multilabel traveling salesman problem. *INFORMS Journal on Computing*, 24 (4):554–564, nov 2012. doi:10.1287/ijoc.1110.0476.

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

G. Kirlik and S. Sayın. Computing the nadir point for multiobjective discrete optimization problems. *Journal of Global Optimization*, 62(1):79–99, aug 2014. doi:10.1007/s10898-014-0227-6.

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

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

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

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

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

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

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

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

J. Sylva and A. Crema. A method for finding the set of non-dominated vectors for multiple objective integer linear programs. *European Journal of Operational Research*, 158(1):46 – 55, 2004. doi:10.1016/S0377-2217(03)00255-8.

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

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

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

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

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

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