Bi-objective branch–and–cut algorithms: Applications to the single source capacitated facility location problem

Sune Lauth Gadegaard[a,*], Lars Relund Nielsen[a], Matthias Ehrgott[b]

[a]*Department of Economics and Business Economics, School of Business and Social Sciences, Aarhus University, Fuglesangs Allé 4, DK-8210 Aarhus V, Denmark.*
[b]*Department of Management Science, Lancaster University, Lancaster LA1 4YX, UK*

Abstract

Most real–world optimization problems are of a multi–objective nature, involving objectives which are conflicting and incomparable. Solving a multi–objective optimization problem requires a method which can generate the set of rational compromises between the objectives. In this paper, we propose two distinct bound set based branch–and–cut algorithms for bi–objective combinatorial optimization problems, based on implicitly and explicitly stated lower bound sets, respectively. The algorithm based on explicitly given lower bound sets computes for each branching node a lower bound set and compares it to an upper bound set. The implicit bound set based algorithm, on the other hand, fathoms branching nodes by generating a single point on the lower bound set for each local nadir point. We outline several approaches for fathoming branching nodes and we propose an updating scheme for the lower bound sets that prevents us from solving the bi–objective LP–relaxation of each branching node. To strengthen the lower bound sets, we propose a bi–objective cutting plane algorithm that dynamically adjusts the weights of the objective functions such that different parts of the feasible set are strengthened by cutting planes. In addition, we suggest an extension of the branching strategy "Pareto branching". Extensive computational results obtained for the bi–objective single source capacitated facility location problem prove the effectiveness of the algorithms.

Keywords: bi–objective branch–and–cut, bi–objective optimization, combinatorial optimization, branch–and–cut.

1. Introduction

A general definition of a multi–objective decision problem with k objectives can be given as follows: Let $\mathcal{X}$ be a set of decisions available to a decision maker, and let $x \in \mathcal{X}$ be a specific decision. Now assume that the decision x can be quantified by real valued functions $f_i : \mathcal{X} \to \mathbb{R}$, $i = 1, \dots, k$. The multi–objective optimization problem then consists in finding all decisions constituting *rational compromises* (Pareto–optimal solutions), with respect to the objective functions f_i, $i = 1, \dots, k$. When the decisions $\mathcal{X}$ are implicitly given by a set of constraints, the problem belongs to the set of multi–objective programming problems, and if the objective functions f_i and the constraints are all

[*]Corresponding author.
Email address: sgadegaard@econ.au.dk (Sune Lauth Gadegaard)

linear, the problem is denoted a multi–objective linear programming problem. In this paper, we will consider the problem where all variables are restricted to be either zero or one and for which only two linear objective functions are present. Such a problem is usually named a *bi–objective combinatorial optimization* (BOCO) problem.

In the following paragraphs we provide an overview of some of the most important contributions to the development of branch–and–bound algorithms for general BOCO problems. A schematic overview is also given in Table 1. In the column headed "x" we have illustrated the domain of the variables where we use the convention that $\mathbb{B} = \{0, 1\}$. Furthermore, when the problem is a *mixed integer* BOCO, we write MI$\mathcal{S}$ where ($\mathcal{S}$) is the discrete domain for the integer restricted variables. The "f" column shows the domain of the objective functions, that is, "$(\mathbb{R}, \mathbb{Z})$" means that the first objective maps to the real numbers and the second is restricted to map into the integers. Furthermore, if the corresponding paper treats problems with more than two objectives, we simply write "$(\mathbb{R}, \ldots, \mathbb{R})$".

Surprisingly little research has been devoted to branch–and–bound algorithms for general BOCO problems although many problems can be fitted into this framework, for example the bi-objective knapsack problem (Ulungu and Teghem, 1997), the bi–objective assignment problem (Przybylski et al., 2008; Pedersen et al., 2008), bi–objective facility location problems (Fernandez and Puerto, 2003), and the bi–objective TSP (Bérubé et al., 2009).

However, more effort has been put into the development of branch–and–bound algorithms for BOCO problems lately. Klein and Hannan (1982) propose what is probably the first branch–and–bound algorithm for BOCO problems. During the eighties and nineties, only very few researchers followed up on this idea, as examples we mention Kiziltan and Yucaoğlu (1983), Ulungu and Teghem (1997), Ramos et al. (1998) and Visée et al. (1998) (although the latter three are problem specific). Since the turn of the millennium more attention has been brought to this solution approach and even generalizations where both integer and continuous variables are allowed (mixed integer BOCO problems) were considered. Mavrotas and Diakoulaki (1998) are among those who develop a branch-and-bound algorithm for BOCO problems. They develop a depth first branch–and–bound algorithm capable of finding all non–dominated outcome vectors of a mixed integer BOCO problem. Whenever a leaf node is reached (that is, when all integer variables have been fixed), the resulting bi–objective linear program is solved. The resulting outcome vectors are compared to solutions in the incumbent set; all outcomes in the set of yet non–dominated outcomes dominated by the outcome vector of a new solution are removed and only non–dominated outcomes are added. A node is fathomed in the branching tree if it is infeasible or if the *ideal point* of the node is dominated by a point in the set of yet non–dominated points. Later, Mavrotas and Diakoulaki (2005) published a number of improvements. The algorithm was adapted to binary combinatorial optimization problems (all variables are either zero or one) and was applied to multi–objective, multi–dimensional knapsack problems in Florios et al. (2010).

Unfortunately the algorithm proposed in the two papers by Mavrotas and Diakoulaki (1998, 2005) might return dominated solutions. The issue originates in the fact that not only the extreme points of the non–dominated frontier found at the leaf nodes might be non–dominated. Also the line segments joining these need to be considered. This issue is addressed in Vincent (2009) and corrected for the bi–objective case by Vincent et al. (2013). In the latter paper a number of lower bound sets are introduced and promising results are reported with up to 60 constraints and 60 variables, of which 30 are binary.

Sourd et al. (2006) and Sourd and Spanjaard (2008) develop a branch–and–bound framework

Table 1: Overview of the bi–objective branch–and–bound algorithms proposed in the literature. All approaches has been converted to the minimization case for better comparison.

Reference	Node selection[1]	Lower bound[2]	Upper bound	Cuts	Branching[3]	x	f_i	Notes
Klein and Hannan (1982)	—	UP	Incumbent set	—	Variable fixing	$\mathbb{N}_0$	$(\mathbb{Z},\dots,\mathbb{Z})$	Propose to use post–optimality techniques to solve a series of integer programs.
Kiziltan and Yucaoğlu (1983)	DF	UP	Incumbent set	—	Variable fixing	$\mathbb{B}$	$(\mathbb{R},\dots,\mathbb{R})$	Probing on variables is applied in each node of the branching tree.
Ulungu and Teghem (1997)	DF	UP	Incumbent set	—	Variable fixing	$\mathbb{B}$	$(\mathbb{Z},\mathbb{Z})$	Adapt the methods presented in Martello and Toth (1990) to a multi–objective framework.
Ramos et al. (1998)	DF	UP	Incumbent set	—	Variable fixing	$\mathbb{B}$	$(\mathbb{R},\mathbb{R})$	Pure branch–and–bound. Feasible solutions are found as leaf nodes, no heuristics.
Visée et al. (1998)	DF	UP	Incumbent set	—	Variable fixing	$\mathbb{B}$	$(\mathbb{Z},\mathbb{Z})$	Use the two phase method and employ the method of Ulungu and Teghem (1997) in the resulting triangles.
Mavrotas and Diakoulaki (1998)	DF	UP	Incumbent set	—	Variable fixing	MI($\mathbb{B}$)	$(\mathbb{R},\dots,\mathbb{R})$	Allow continuous variables. Problematic, as some dominated solutions might be considered non–dominated.
Mavrotas and Diakoulaki (2005)	DF	UP	Incumbent set	—	Variable fixing	MI($\mathbb{B}$)	$(\mathbb{R},\dots,\mathbb{R})$	Computationally improve the solution of the LP–relaxation. Add a final dominance test. Still problematic.
Sourd and Spanjaard (2008)	DF	HS	Incumbent set	—	Variable fixing	$\mathbb{B}$	$(\mathbb{R},\mathbb{R})$	Propose to use hypersurfaces as lower bound set.
Florios et al. (2010)	DF	UP	Incumbent set	—	Variable fixing	$\mathbb{B}$	$(\mathbb{R},\mathbb{R})$	Application of the algorithm proposed in Mavrotas and Diakoulaki (1998, 2005).
Vincent et al. (2013)	DF	UP	Incumbent set	—	Variable fixing	MI($\mathbb{B}$)	$(\mathbb{R},\mathbb{R})$	Correct and improve the algorithm of Mavrotas and Diakoulaki (1998). Characterize the non–dominated frontier of a mixed integer BOCO problem.
Stidsen et al. (2014)	—	S–LP	Incumbent set	—	Variable fixing, PB, SL	MI($\mathbb{B}$)	$(\mathbb{Z},\mathbb{R})$	Introduce Pareto branching and slicing of the outcome space. One objective must be integer valued.
This paper	BB	S–LP BO–LP	Incumbent set	Yes	Variable fixing, extended PB	$\mathbb{B}$	$\mathbb{Z}$	Introduces cutting plane algorithm. Proposes updating of lower bound sets. Extends PB.

[1] Node selection (DF: Depth first, BB: Best bound) [2] Lower bound (HS: Hyper surface, UP: Utopian or ideal point, S–LP: LP–relaxation of scalarized problem, BO–LP: Bi–Objective LP–relaxation) [3] Branching: (PB: Pareto branching, SL: Slicing)

where the branching part is identical to a single objective branch–and–bound algorithm. However, the bounding part is performed via a set of points rather than the single ideal point. The current node can be discarded if a hypersurface separates the set of feasible solutions in the subproblem from the incumbent set (the set of non–dominated points). Sourd and Spanjaard use a rather sophisticated problem–specific hypersurface and obtain promising experimental results for the bi–objective spanning tree problem.

Very recently Stidsen et al. (2014) introduced the concept of *Pareto branching* where branching is performed in outcome space. Also, they proposed *slicing* the outcome space and thereby obtaining better upper bounds for fathoming nodes in the branching tree. Promising test results are provided for a range of bi–objective combinatorial optimization problems.

Although several novel and efficient approaches have been proposed in the past, none of these takes advantage of the lower bound set available from the bi–objective LP–relaxation. Furthermore, none of the previously mentioned algorithms incorporates cutting planes, even though effective separation routines have resulted in a significant speedup for single objective problems. Therefore, we propose two novel bound set based branch–and–cut algorithms for bi–objective linear combinatorial optimization problems. The algorithms rely on either explicitly or implicitly given lower bound sets obtained from the bi–objective LP–relaxation. To summarize, the main contributions of this paper are as follows

1. We propose a bi–objective cutting plane algorithm which dynamically changes weights of the objectives in order to approximate the best possible lower bound set obtainable from the LP–relaxation.
2. We develop a simple updating scheme for explicit lower bound sets that reduces the number of bi–objective LPs that need to be solved.
3. We propose a simple method for implicitly describing the lower bound set obtained from the bi–objective LP–relaxation.
4. The Pareto branching strategy is strengthened to what we call extended Pareto branching.
5. In total 8 different branch–and–cut implementations are evaluated on the bi–objective single source capacitated facility location problem, which, to the best of our knowledge, is solved for the first time in the literature.

The remainder of this paper is organized as follows: Section 2 gives the basic definitions of bi–objective optimization. Section 3 starts with a short theoretical description of a generic bi–objective branch–and–cut algorithm and afterwards we describe the main components of the branch–and–cut algorithm in detail. Finally, different implementations of the algorithm developed in the paper are tested in Section 4.

2. Preliminaries

The focus of this section will be on a generic linear *bi-objective combinatorial optimization* (BOCO) problem of the form

$$\min\{Cx \, : \, x \in \mathcal{X}\} \tag{1}$$

where $C = (c^1, c^2)$ is a $2 \times n$ dimensional matrix with all entries being integral and the *feasible set* is defined by $\mathcal{X} = \{x \in x \in \{0,1\}^n \, : \, Ax \leqq b\}$. The set $\mathcal{X}$ of feasible solutions is also referred to as the feasible set in *decision space* and the image of $\mathcal{X}$ under the linear mapping C is called the feasible set in *objective space* and is here denoted $\mathcal{Z}$.

To compare vectors in $\mathbb{R}^2$ we adopt the notation from Ehrgott (2005). Let $z^1, z^2 \in \mathbb{R}^2$, then

$$\begin{aligned} z^1 \leqq z^2 &\Leftrightarrow z^1_k \leqq z^2_k, \text{ for } k = 1,2 \\ z^1 \leq z^2 &\Leftrightarrow z^1 \leqq z^2 \text{ and } z^1 \neq z^2 \\ z^1 < z^2 &\Leftrightarrow z^1_k < z^2_k, \text{ for } k = 1,2. \end{aligned}$$

We define the set $\mathbb{R}^2_{\geqq} = \{z \in \mathbb{R}^2 : z \geqq 0\}$ and analogously $\mathbb{R}^2_{\geq}$ and $\mathbb{R}^2_{>}$. Furthermore, given a set $S \subseteq \mathbb{R}^2$ let $S_N = \{s \in S : (\{s\} - \mathbb{R}^2_{\geqq}) \cap S = \{s\}\}$, where $(\{s\} - \mathbb{R}^2_{\geqq}) = \{z \in \mathbb{R}^2 : z = s - r,\ r \in \mathbb{R}^2_{\geqq}\}$. The set S_N is called the *non–dominated set of* S.

The problem (1) does not immediately reveal what an "optimal solution" should be. To clarify this we use the concept of Pareto optimality or efficiency:

Definition 1. A feasible solution $\hat{x} \in \mathcal{X}$ is called Pareto optimal or *efficient* if there does not exist any $x \in \mathcal{X}$ such that $Cx \leq C\hat{x}$. The image $C\hat{x}$ is then called *non-dominated.*

A feasible solution $\hat{x} \in \mathcal{X}$ is called weakly efficient if there does not exist any $x \in \mathcal{X}$ such that $Cx < C\hat{x}$.

Let $\mathcal{X}_E$ denote the set of all efficient solutions. Then the image of $\mathcal{X}_E$ under the linear mapping C is exactly $\mathcal{Z}_N$, that is, $\mathcal{Z}_N = C\mathcal{X}_E$. The set $\mathcal{Z}_N$ is referred to as the set of non–dominated outcomes. A subset $\mathcal{X}^* \subseteq \mathcal{X}_E$ where $C\mathcal{X}^* = \mathcal{Z}_N$ and $Cx \neq Cx'$ for all $x, x' \in \mathcal{X}^*$ will be considered an optimal solution to (1). Note that an optimal solution $\mathcal{X}^*$ is a *set* of efficient solutions.

The sets $\mathcal{X}_E$ and $\mathcal{Z}_N$ need to be further divided into two subsets. An efficient solution $\hat{x} \in \mathcal{X}_E$ is said to be a *supported* efficient solution if there exists a weight $\lambda \in (0,1)$ such that $\lambda c^1\hat{x} + (1-\lambda)c^2\hat{x} \leqq \lambda c^1 x + (1-\lambda)c^2 x$ for all $x \in \mathcal{X}$. The set of supported efficient solutions is denoted $\mathcal{X}_{SE}$. The elements in $\mathcal{X}_{NE} = \mathcal{X}_E \setminus \mathcal{X}_{SE}$ are called *non–supported* efficient solutions. Analogously, the set $\mathcal{Z}_N$ is partitioned into two subsets, namely $\mathcal{Z}_{SN} = C\mathcal{X}_{SE}$ and $\mathcal{Z}_{nN} = C\mathcal{X}_{NE}$.

3. Bi–objective bound set based branch–and–cut

A branch–and–cut framework provides a very successful standard method for solving single objective combinatorial optimization problems (see e.g. Nemhauser and Wolsey (1988) or Martin (1999) for a detailed description). Here, the set of feasible solutions to the optimization problem is partitioned into disjoint subproblems which can be displayed in a tree–structure where each *node* represents a *subproblem.* We say that a branching node is *fathomed* if it has been proven that the subproblem corresponding to that branching node cannot contain solutions improving the current best solution or if the corresponding subproblem is infeasible. The algorithm keeps a set H of *active* nodes, that have not been fathomed. A specific active branching node is denoted η. Let $\mathcal{X}(\eta)$ denote the set of feasible solutions of the subproblem corresponding to branching node η. That is, the solutions in $\mathcal{X}$ satisfying all branching constraints added on the unique path from the root node to the branching node η. Furthermore, we let $\underset{\sim}{\mathcal{X}}$ ($\underset{\sim}{\mathcal{X}}(\eta)$) denote the set $\mathcal{X}$ ($\mathcal{X}(\eta)$) with all integrality constraints removed.

For single objective optimization problems only a single optimal solution value exists, say $z^* \in \mathbb{R}$. Thus, upper and lower bounds on z^* are given by numbers $u, l \in \mathbb{R}$ satisfying $l \leqq z^* \leqq u$. To adapt a branch–and–cut framework to BOCO problems we need to consider bounds on the set $\mathcal{Z}_N$ of non–dominated solution values, hence we naturally need to extend the concept of bounds to *bound sets*. We use the definition of bound sets given in Ehrgott and Gandibleux (2007), stated for the bi–objective case below.

Definition 2 (Bound sets)**.** Lower and upper bound sets are defined as follows:

1. A *lower bound set* on $\mathcal{Z}_N$ is a subset $L \subseteq \mathbb{R}^2$ such that L is an $\mathbb{R}^2_{\geqq}$–closed and $\mathbb{R}^2_{\geqq}$–bounded set with $L = L_N$ such that
$$\mathcal{Z}_N \subseteq (L + \mathbb{R}^2_{\geqq}).$$
Given two lower bound sets L_1 and L_2 we say that L_1 *dominates* L_2 if $L_1 \subseteq L_2 + \mathbb{R}^2_{\geqq}$. If furthermore $L_1 + \mathbb{R}^2_{\geqq} \neq L_2 + \mathbb{R}^2_{\geqq}$ we say that L_1 *strictly* dominates L_2.

2. An *upper bound set* on $\mathcal{Z}_N$ is a subset $U \subseteq \mathbb{R}^2$ such that U is an $\mathbb{R}^2_{\geqq}$–closed and $\mathbb{R}^2_{\geqq}$–bounded set with $U = U_N$ such that
$$\mathcal{Z}_N \subseteq \operatorname{cl}\left(\mathbb{R}^2 \setminus (U + \mathbb{R}^2_{\geq})\right),$$
where $\operatorname{cl}(S)$ denotes the closure of a set $S \subseteq \mathbb{R}^2$.

The lower bound set L is called $\mathbb{R}^2_{\geqq}$*–convex* if the set $(L + \mathbb{R}^2_{\geqq})$ is convex. In this paper we will focus on the $\mathbb{R}^2_{\geqq}$–convex lower bound set available from the non–dominated frontier of the LP-relaxation of the BOCO, that is

$$(C\mathcal{X})_N = \{z \in \mathbb{R}^2 \;:\; z = Cx,\; Ax \leqq b,\; x \in [0,1]^n\}_N.$$

From Definition 2 it is also readily seen that any set of feasible solutions filtered by dominance gives rise to an upper bound set. A $\mathbb{R}^2_{\geqq}$–convex lower bound set and an upper bound set is illustrated in Figure 1(a).

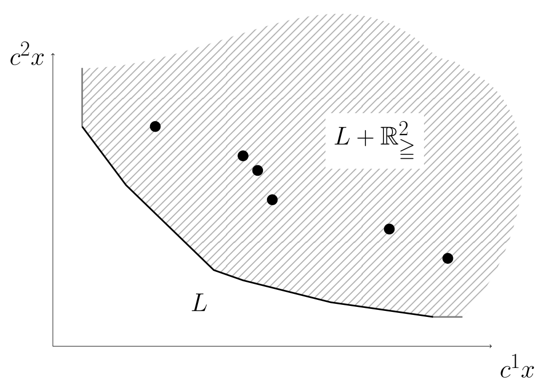
For single objective optimization problems an active node in the branching tree can be fathomed if the subproblem corresponding to the branching node is infeasible or if the lower bound of the subproblem is greater than or equal to the global upper bound. To extend this result to a multi–objective branch–and–cut algorithm, we need the following definition.

Definition 3 (Local nadir point)**.** Let $U = \{z^1, \ldots, z^{|U|}\} \subseteq \mathcal{Z}$ be an upper bound set of feasible points ordered such that $z_1^u < z_1^{u+1}$, for all $z^u, z^{u+1} \in U$. Then the set of *local nadir points* is given by
$$\mathcal{N}(U) = \bigcup_{u=1}^{|U|-1} \{(z_1^{u+1}, z_2^u)\}.$$

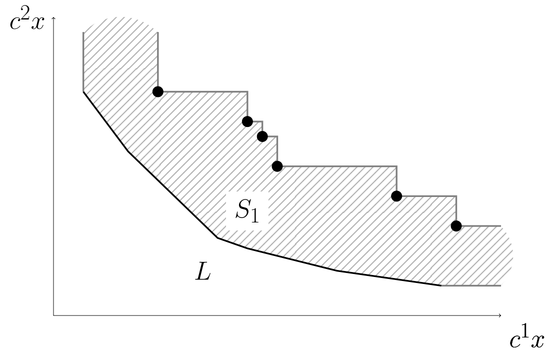
In a multi–objective branch–and–cut algorithm an active node η in the branching tree, can be fathomed if the subproblem corresponding to the branching node is infeasible or if every solution in the subproblem corresponding to η is dominated by at least one solution in the upper bound set U. That is, the *search area* between the lower and upper bound sets must be empty.

The search area may be defined in different ways. As noted by Przybylski et al. (2010), given a branching node η, its feasible points in objective space $\mathcal{Z}(\eta) := C\mathcal{X}(\eta)$, a lower bound set $L(\eta)$ of $\mathcal{Z}(\eta)$, and an upper bound set U we have

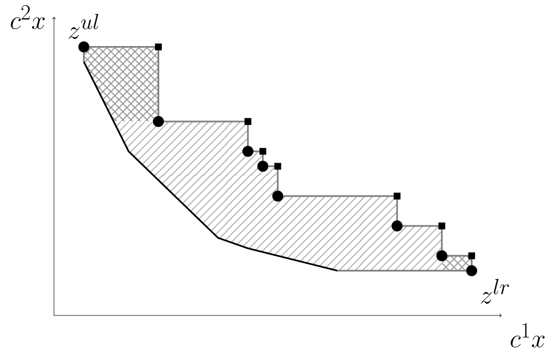
$$(\mathcal{Z}(\eta) \cap \mathcal{Z}_N) \subseteq \left(L(\eta) + \mathbb{R}^2_{\geqq}\right) \setminus \left(U + \mathbb{R}^2_{>}\right) = S_1. \tag{2}$$



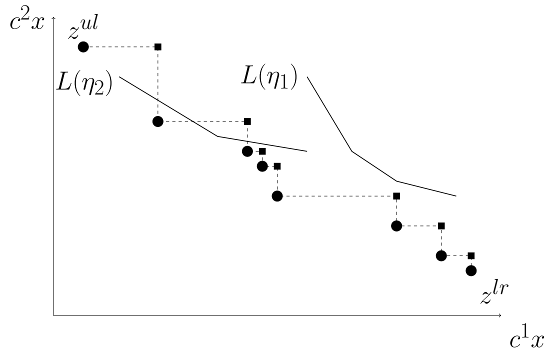
(a) Upper and lower bound sets. The hatched area is $L + \mathbb{R}^2_{\geqq}$.



(b) Search area S_1 defined in (2) (hatched area).



(c) Search area S_2 defined in (3) (hatched and cross-hatched area). Note that if the lexicographic minima are not part of the upper bound set, then the cross-hatched areas are missing.



(d) The node η_1 *can* be fathomed, since no local nadir point is positioned "above" the lower bound set. The node η_2 can not be fathomed.

Figure 1: Illustrations of the search area. Lower bound sets are illustrated with solid black lines, upper bound sets with circles, and local nadir points with squares.

That is, the search area is defined as S_1 (see Figure 1(b) for an illustration). This representation is, however, not that useful in an algorithmic sense. By assuming that the lexicographic minima are part of the upper bound set U, we get the inclusion

$$(\mathcal{Z}(\eta) \cap \mathcal{Z}_N) \subseteq \left(L(\eta) + \mathbb{R}^2_{\geqq}\right) \cap \left(\mathcal{N}(U) - \mathbb{R}^2_{\geqq}\right) = S_2. \tag{3}$$

The search area S_2 is illustrated in Figure 1(c) (hatched and cross-hatched area). Note that if the lexicographic minima are not part of the upper bound set, then (3) does not hold (the cross-hatched areas are missing). Given an upper bound set U including the lexicographic minima and a lower bound set $L(\eta)$ on $\mathcal{Z}(\eta)$, the search for non–dominated points can be restricted to the search area S_2 and a sufficient condition for fathoming a branching node η is

$$\left(L(\eta) + \mathbb{R}^2_{\geqq}\right) \cap \left(\mathcal{N}(U) - \mathbb{R}^2_{\geqq}\right) = \emptyset \tag{4}$$

Moreover, the result can be strengthened further.

Proposition 1. *Given an active branching node η, a lower bound set $L(\eta)$, and an upper bound set U including the lexicographic minima, the branching node η can be fathomed if*

$$\left(L(\eta) + \mathbb{R}^2_{\geqq}\right) \cap \mathcal{N}(U) = \emptyset. \tag{5}$$

Proof. We show that (5) implies (4). Assume that (5) holds true and that $\bar{z} \in \left(L(\eta) + \mathbb{R}^2_{\geqq}\right) \cap \left(\mathcal{N}(U) - \mathbb{R}^2_{\geqq}\right) \neq \emptyset$. Then $\bar{z} \in (\mathcal{N}(U) - \mathbb{R}^2_{\geqq}) \setminus \mathcal{N}(U)$ implying there exists a $\tilde{z} \in \mathcal{N}(U)$ such that $\tilde{z} \in \{\bar{z}\} + \mathbb{R}^2_{\geqq}$. Since $\bar{z} \in L(\eta) + \mathbb{R}^2_{\geqq}$ then so is $\tilde{z}$. This contradicts the starting assumption. □

An illustration of the proposition is given in Figure 1(d). Here the branching node η_1 can be fathomed whereas node η_2 cannot.

Based on the above results a general bi–objective branch–and–cut framework can now be described as given in Algorithm 1. The algorithm is initialized by setting the upper bound set equal to the lexicographic minima of both orders of objectives in Step 0 and an active node is chosen in Step 1. At each node, cuts can be added (Step 2) in order to strengthen the lower bound set obtained in Step 3. If a feasible solution can be obtained from the node it might be a non–dominated solution, and the upper bound set is therefore updated in Step 4. After obtaining a lower bound set and updating the upper bound set, Step 5 is used to test whether the branching node can be fathomed. If not, the branching node is split into several new disjoint *child nodes* (Step 6). Note that Algorithm 1 implicitly enumerates all solutions to the BOCO problem 1 implying that when $H = \emptyset$, the set of non–dominated outcomes has been found. Furthermore, as the number of solutions to the BOCO problem is finite, Algorithm 1 terminates in finite time if the individual steps can be performed in finite time. In the following subsections we elaborate on Steps 2 through 6.

3.1. Step 2 - Adding cutting planes

A major challenge when designing algorithms for BOCO problems is to find a way to efficiently utilize the numerous methods and strategies available for the single objective versions of the BOCO problems. In this section, we propose a way to utilize cutting planes to reduce the “gap” between

```
Step 0 Initialize the upper bound set $U$ and set $H$ equal to the root node.
Step 1 If $H = \emptyset$, then return $U = \mathcal{Z}_N$ and stop; otherwise select an active branching node
       $\eta \in H$.
Step 2 Add cutting planes.
Step 3 Obtain a lower bound set $L(\eta)$ of node $\eta$.
Step 4 Update the upper bound set $U$.
Step 5 If the node $\eta$ can be fathomed, go to Step 1.
Step 6 Perform branching on $\eta$. Go to Step 1.
```

Algorithm 1: Generic multi–objective branch–and–cut algorithm based on bound sets.

the lower bound set provided by the bi–objective LP–relaxation and the set of non–dominated outcomes to the BOCO problem (1).

In a single objective branch–and–cut algorithm, cutting planes are used before a branch–and–bound algorithm is started, as a way to improve the lower bound. Or seen from another perspective cuts are added to generate a tighter representation of the convex hull of integer solutions *in the direction of the objective function.* In case of a BOCO problem there is not a single direction of the objective function, and it is not obvious in which part of the polyhedron corresponding to the LP-relaxation, cutting planes would be most beneficial.

Therefore, we solve the LP–relaxation of a weighted sum scalarization of the BOCO problem and add cutting planes (in decision space) for different weights. The goal is to generate a relaxation of the BOCO problem which provides a lower bound set as close to $\operatorname{conv}(\mathcal{Z}_N)_N$ as possible. Note, that the strongest $\mathbb{R}^2_{\geqq}$–convex lower bound set is

$$\operatorname{conv}(\mathcal{Z}_N)_N = \{z \in \mathbb{R}^2 \,:\, z = Cx,\ x \in \operatorname{conv}(\mathcal{X})\}_N.$$

From this description, we see that by approximating $\operatorname{conv}(\mathcal{X})$ using cutting planes, we also approximate the strongest possible $\mathbb{R}^2_{\geqq}$–convex lower bound set without having to solve a series of integer programming problems.

An overview of the cutting plane algorithm is given in Algorithm 2. The algorithm starts by choosing a search direction (or weight) in Step 2.0. When the weight has been chosen, an ordinary cutting plane algorithm is used to separate cutting planes in the part of the decision space identified by the search direction λ (Step 2.1 and Step 2.2). In Step 2.3 a stopping criterion for the cutting plane algorithm is checked, allowing for multiple *rounds* of cuts. Finally, in Step 2.4, we check if a new search direction should be chosen or not. The algorithm described in Algorithm 2 distinguishes itself from a single objective cutting plane algorithm only by the outer loop where different search directions are used. This means that problem specific cutting planes can be used in Step 2.2, if effective separation algorithms are known.

The algorithmic framework in Algorithm 2 leaves two obvious ways of choosing the search direction in Step 2.1: An *a priori* and a *dynamic* approach. An example of an a priori approach

```
Step 2.0 Choose weight $\lambda \in (0,1)$.
Step 2.1 Solve the weighted sum scalarization of the BOCO problem
             $\min\{(\lambda c^1 + (1-\lambda)c^2)x \,:\, x \in \underline{\mathcal{X}}\}$,
         and let $x^*$ be an optimal solution.
Step 2.2 If possible, separate $x^*$ using cutting planes and append the cutting planes to the
         description of $\underline{\mathcal{X}}$.
Step 2.3 If we should add further cuts (e.g. if new cuts were added in Step 2.2), go to Step 2.1.
Step 2.4 If we should apply a new search direction $\lambda$, go to Step 2.1; otherwise stop.
```

Algorithm 2: Step 2 of Algorithm 1.

would be to pick the values for λ in the set $\{\frac{1}{k}, \frac{2}{k}, \dots, \frac{k-1}{k}\}$, for some number $k > 0$, whereas a dynamic strategy would be to chose λ based on the previous iteration of the algorithm. In this paper we have implemented a dynamic updating scheme based on a modification of the so–called *Non–Inferior Set Estimation* framework proposed by Cohon (1978), Aneja and Nair (1979), and Dial (1979).

3.2. Step 3 - Obtaining a lower bound set

In this section, we describe how we derive lower bound sets of the current branching node η in Step 3 of Algorithm 1. Consider an active branching node η and let

$$L^C(\eta) = (C\underline{\mathcal{X}}(\eta))_N$$

denote the lower bound set equal to the set of non–dominated outcome vectors of the bi–objective LP–relaxation of the current node. One approach to obtaining lower bound sets would then be to solve the bi–objective LP–relaxation in each branching node and use Proposition 1 to test if the node can be fathomed. However, it may be computationally expensive to solve a bi–objective LP and checking condition (5) in Proposition 1 at every branching node. Therefore, we only want to solve the bi–objective LP at branching nodes where there is a possibility that condition (5) holds (the search area S_2 defined in (2) is empty). Given $\lambda \in (0,1)$, let

$$\Lambda^\lambda(\eta) = \min\{(\lambda c^1 + (1-\lambda)c^2)x \,:\, x \in \underline{\mathcal{X}}(\eta)\} \tag{6}$$

denote the optimal solution value of the *λ–scalarized LP–relaxation* of the node η and

$$x^\lambda(\eta) \in \arg\min\{(\lambda c^1 + (1-\lambda)c^2)x \,:\, x \in \underline{\mathcal{X}}(\eta)\},$$

an optimal solution to (6). The following proposition gives sufficient conditions to ensure that S_2 is non-empty..

Proposition 2. *Consider upper bound set U and lower bound set $L(\eta) = L^C(\eta)$ and assume that the solution $x^\lambda(\eta)$ satisfies*

$$Cx^\lambda(\eta) \leq z^n$$

for some $z^n \in \mathcal{N}(U)$. Then

$$\left(L(\eta) + \mathbb{R}^2_{\geqq}\right) \cap \mathcal{N}(U) \neq \emptyset.$$

Proof. Suppose there exists a local nadir point $z^n \in \mathcal{N}(U)$ such that $Cx^\lambda(\eta) \leq z^n$. Since $Cx^\lambda(\eta) \in L(\eta)$, we have that $z^n \in L(\eta) + \mathbb{R}^2_{\geqq}$ implying $\left(L(\eta) + \mathbb{R}^2_{\geqq}\right) \cap \mathcal{N}(U) \neq \emptyset$. □

Proposition 2 suggests to always solve a λ–scalarized LP before solving the bi–objective LP relaxation. If Proposition 2 holds, then there is no reason to solve the bi–objective LP, since the branching node η cannot be fathomed. Furthermore, solving the λ–scalarized LP provides us with a lower bound set

$$L^\lambda(\eta) = \{z \in \mathbb{R}^2 \; : \; \lambda z_1 + (1-\lambda) z_2 = \Lambda^\lambda(\eta)\},$$

which may be used to find a lower bound set $L(\eta)$ at branching node η as described in Proposition 3.

Proposition 3. *Given an active branching node η and parent node η_0 with lower bound set $L(\eta_0)$, the set*

$$L(\eta) = \left(\left(L(\eta_0) + \mathbb{R}^2_{\geqq}\right) \cap \left(L^\lambda(\eta) + \mathbb{R}^2_{\geqq}\right)\right)_N,$$

is a lower bound set of branching node η dominating both $L(\eta_0)$ and $L^\lambda(\eta)$.

Proof. Obviously, $L^\lambda(\eta)$ is a lower bound set for η, and by relaxation so is $L(\eta_0)$. The rest follows from Proposition 2 in Ehrgott and Gandibleux (2007). □

Determining the set $L(\eta)$ defined in Proposition 3 is straightforward. Assume that for branching node η_0 the lower bound set $L(\eta_0)$ is represented by an ordered list of extreme points, $\{\underline{z}^1, \ldots, \underline{z}^L\}$, with $\underline{z}_1^l < \underline{z}_1^{l+1}$. The updating is then simply done by first finding two pairs of points $(\underline{z}^{l_1}, \underline{z}^{l_1+1})$ and $(\underline{z}^{l_2}, \underline{z}^{l_2+1})$ satisfying

$$\lambda \underline{z}_1^{l_1} + (1-\lambda)\underline{z}_2^{l_1} \geqq \Lambda^\lambda(\eta) > \lambda \underline{z}^{l_1+1} + (1-\lambda)\underline{z}_2^{l_1+1}$$

and

$$\lambda \underline{z}_1^{l_2} + (1-\lambda)\underline{z}_2^{l_2} < \Lambda^\lambda(\eta) \leqq \lambda \underline{z}^{l_2+1} + (1-\lambda)\underline{z}_2^{l_2+1},$$

as illustrated in Figure 2. Having determined these two pairs of points, we simply calculate the intersection point between the straight line defined by the two points $\underline{z}^{l_1}$ and $\underline{z}^{l_1+1}$ (respectively $\underline{z}^{l_2}$ and $\underline{z}^{l_2+1}$) and the line defined by the lower bound set $L^\lambda(\eta)$. The two intersection points, say z^{I_1} and z^{I_2}, are inserted in the list and we obtain the new lower bound set $L(\eta)$ defined by

$$L(\eta) = \text{conv}(\{\underline{z}^1, \ldots, \underline{z}^{l_1}, z^{I_1}, z^{I_2}, \underline{z}^{l_2+1}, \ldots, \underline{z}^L\})_N.$$

Step 3 can now be specified in Algorithm 3. We are interested in solving the bi–objective LP–relaxation as few times as possible as long as the updated lower bound set is of sufficient quality. We use the heuristic rule that if the branching performed at the parent node was performed in objective space (see Section 3.5), the bi–objective LP–relaxation is calculated since we want new lower bound sets for different regions in the objective space (Step 3.0). Otherwise we solve the

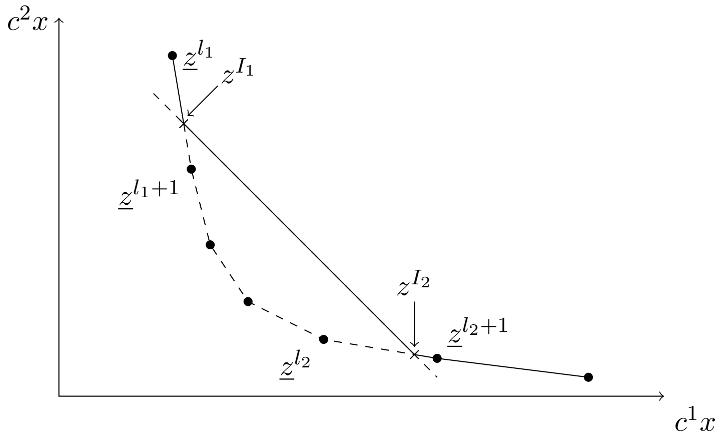


Figure 2: Updating the lower bound set using Proposition 3.

Step 3.0 If the branching at the parent node η_0 has been performed in objective space, solve the LP–relaxation and return $L^C(\eta)$.

Step 3.1 Solve the λ–scalarized LP and find $L(\eta)$ defined in Proposition 3.

Step 3.2 If $L(\eta)$ strictly dominates $L(\eta_0)$, then solve the LP–relaxation and return $L^C(\eta)$; otherwise return $L(\eta)$.

Algorithm 3: Step 3 of Algorithm 1.

```
Step 5.0  If the subproblem corresponding to node $\eta$ is infeasible, fathom $\eta$ and go to Step 1.
Step 5.1  If Proposition 2 holds, go to Step 6.
Step 5.2  If Proposition 1 holds, go to Step 1.
```

Algorithm 4: Step 5 of Algorithm 1.

λ–scalarized LP (Step 3.1) and if the lower bound set of the parent node is strictly dominated by $L(\eta)$ found in Proposition 3 (all the extreme points of $L(\eta_0)$ lie below $L^\lambda(\eta)$), then we resolve the bi–objective LP–relaxation.

3.3. Step 4 - Update the upper bound set

Throughout the branch–and–bound process an upper bound set, U, of feasible points in objective space filtered by dominance is maintained and ordered such that $z_1^u < z_1^{u+1}$ for all $z^u, z^{u+1} \in U$. In Step 0 of Algorithm 1 U is initialized with the two lexicographic minima. Whenever a feasible solution is found, the outcome vector of the solution is inserted into the upper bound set U and the augmented set is filtered by dominance. In this way, the set U is constantly improving and will at any time form an upper bound set. Note that solutions can be obtained using heuristics from all active branching nodes even though the node is not a leaf node (all integer variables are fixed).

3.4. Step 5 - Bound fathoming

Algorithm 4 checks if an active node η in the branching tree, can be fathomed. If the subproblem corresponding to the branching node is infeasible, we fathom η and pick a new active node (Step 5.0); otherwise Proposition 2 is used as a preliminary check for not fathoming the node (Step 5.1). Finally, Proposition 1 is checked in Step 5.2. Proposition 1 can be checked using both an explicit and an implicitly given lower bound set $L(\eta)$. Below we describe three ways of testing Proposition 1.

3.4.1. Bound fathoming using an explicit lower bound set and LP

Assume that a lower bound set $L(\eta)$ is stored as an explicit set using the extreme points $\{\underline{z}^1, \dots, \underline{z}^L\}$ of $L(\eta)$. Note that when $L(\eta)$ is $\mathbb{R}^2_{\geqq}$–convex, the set $L(\eta) + \mathbb{R}^2_{\geqq}$ forms an unbounded convex polygonal domain in $\mathbb{R}^2$, and the verification of the condition in Proposition 1 amounts to verifying if any of the points in $\mathcal{N}(U)$ lie in this convex polyhedron. A simple, and straightforward way of performing this bound fathoming check is to solve the linear program

$$
\begin{aligned}
Z(z^n) = \min\ & s_1 + s_2 \\
\text{s.t.:}\ & \sum_{l=1}^{L} \underline{z}_1^l \lambda_k - s_1 \leq z_1^n \\
& \sum_{l=1}^{L} \underline{z}_2^l \lambda_k - s_2 \leq z_2^n \\
& \sum_{l=1}^{L} \lambda_l = 1 \\
& \lambda_l, s_1, s_2 \geq 0,\ \forall l = 1, \ldots, L
\end{aligned}
\tag{7}
$$

for all $z^n \in \mathcal{N}(U)$. If $Z(z^n) > 0$ for all $z^n \in \mathcal{N}(U)$, the node η can be fathomed based on bounding. Note that the linear programs (7) only have three constraints, leading to linear programs which can be solved very quickly. Furthermore, for two different local nadir points z^n and $z^{n'}$, the linear programs (7) differs in the right hand sides only. This means that very few dual simplex iterations are usually needed in order to resolve these linear programs.

3.4.2. Bound fathoming using an explicit lower bound set and a point–in–polygon algorithm

Let $\{\underline{z}^1, \ldots, \underline{z}^L\}$ denote the extreme points of $L(\eta)$ and note that

$$
\mathcal{Z}_N \subseteq \left(\{(z_1^{lr}, z_2^{ul})\} - \mathbb{R}^2_{\geqq} \right),
$$

where $z^{ul} = \operatorname{lex}\min\{(c^1x, c^2x) : x \in \mathcal{X}\}$ and $z^{lr} = \operatorname{lex}\min\{(c^2x, c^1x) : x \in \mathcal{X}\}$ are the two lexicographic minima. Hence Proposition 1 does not hold if a local nadir point is in the polygon given by

$$
\operatorname{conv}(\{(\underline{z}_1^1, z_2^{ul}), \underline{z}^1, \ldots, \underline{z}^L, (z_1^{lr}, \underline{z}_2^L), (z_1^{lr}, z_2^{ul})\}). \tag{8}
$$

This means that the condition in Proposition 1 can be tested by calling, for each local nadir point, a *point–in–polytope* algorithm which tests for inclusion in the polygon (8). As soon as the algorithm declares that a local nadir point is within the polygon, we know the branching node cannot be fathomed. Fortunately, much research has gone into point–in–polytope (PIP) algorithms and we refer the interested reader to the computational study by Schirra (2008) of the reliability and speed of a number of different algorithmic approaches for the PIP problem.

3.4.3. Bound fathoming using an implicit lower bound set and LP

The bound fathoming can also be done without explicitly maintaining a lower bound set. Note that the condition in Proposition 1 asks if any point on the lower bound set dominates a local nadir point and that the lower bound set of the branching node η we are using is $(C\mathcal{X}(\eta))_N$. Deciding if there exists a point $\underline{z} \in (C\underline{\mathcal{X}}(\eta))_N$ which (strictly) dominates a local Nadir point $z^n \in \mathcal{N}(U)$ is equivalent to having $\tilde{Z}(z^n) = 0$ where

$$\begin{aligned} \tilde{Z}(z^n) = \min\ & s_1 + s_2 \\ \text{s.t.:}\ & c^1x - s_1 \leq z_1^n \\ & c^2x - s_2 \leq z_2^n \\ & x \in \underline{\mathcal{X}}(\eta) \\ & s_1, s_2 \geq 0. \end{aligned} \tag{9}$$

Note that an optimal solution (x^*, s^*) to the linear program (9) either has $\tilde{Z}(z^n) = 0$ implying $Cx^* \leq z^n$ (that is, there is a solution to the LP-relaxation which dominates z^n) or we have $\tilde{Z}(z^n) = s_1^* + s_2^* > 0$. This leads to Proposition 4.

Proposition 4. *The branching node η can be fathomed if the optimal solution value $\tilde{Z}(z^n)$ of the program* (9) *is strictly positive for all $z^n \in \mathcal{N}(U)$.*

This implicit approach eliminates the need for generating the entire efficient frontier of the LP-relaxation and only one linear program needs to be solved for each local nadir point. The approach might then have its merits for problems with few non–dominated outcomes compared to the number of extreme points of $(C\underline{\mathcal{X}}(\eta))_N$, but this is not a trivial matter to decide a priori. A possible drawback compared to generating the complete set $(C\underline{\mathcal{X}}(\eta))_N$ is that the updating scheme described in Proposition 3 no longer applies, implying that additional linear programming problems need to be solved at each branching node. Furthermore, the program (9) might be large and solving it may consequently be rather time consuming. As the program (9) must potentially be solved for several local nadir points, this might lead to prohibitive computation times.

3.5. Step 6 - Performing branching

As the set of feasible solutions does not differ compared to a single objective combinatorial optimization problem, the branching rules devised for these problems can be applied. However, much information can be gained by utilizing the definition of an efficient solution and its non–dominated outcome vector. As mentioned in Section 3.2, the bi–objective LP–relaxation is not solved at each branching node. However, the weighted sum scalarization

$$\min\{(\lambda c^1 + (1-\lambda)c^2)x \ : \ x \in \underline{\mathcal{X}}(\eta)\} \tag{10}$$

is solved. Let $\underline{x}(\eta)$ be an optimal solution to problem (10), and let $\underline{z}(\eta) = C\underline{x}(\eta)$ be the corresponding outcome vector. First, we outline the very effective branching strategy proposed in Stidsen et al. (2014) called *Pareto branching* (PB). PB is based on the observation that if $z \leqq \underline{z}(\eta)$ for all z in the *ordered* sublist $\{\bar{z}^{u_1}, \dots, \bar{z}^{u_K}\} \subseteq U$ of the current upper bound set, then the branching node η can be split by the disjunction

$$c^1x \leq \bar{z}_1^{u_1} - 1 \quad \vee \quad c^2x \leq z_2^{u_K} - 1.$$

We note, that in our implementation we update the upper bound set U *before* branching. This means, that if a node results in an integer feasible solution, the outcome vector of this solution is part of the upper bound set when branching is performed. Therefore, if a branching node results in an integer feasible solution, Pareto branching can always be performed and there is no need to add the weaker so–called no–good inequalities

$$\sum_{i:\underline{x}(\eta)_i=0} x_i + \sum_{i:\underline{x}(\eta)_i=1} (1-x_i) \geq 1,$$

used in ranking based two–phase methods.

The idea of PB can be expanded to *extended Pareto branching* (EPB). From the definition of the search area given in (3), it is evident that non–dominated outcomes can only exist in the set

$$\bigcup_{z^n \in \left(L(\eta)+\mathbb{R}^2\right)\cap\mathcal{N}(U)} \left(z^n - \mathbb{R}^2_{\geqq}\right).$$

Before making a branching decision, we already check whether local nadir points exist in the set $\left(L(\eta)+\mathbb{R}^2_{\geqq}\right)$ (see Proposition 1 and Proposition 4),and therefore we can simply create a child node for each local Nadir point found. Note that this split of the branching node η might not separate the current LP–solution as we may have that

$$Cx(\eta) \in \bigcup_{z \in \left(L(\eta)+\mathbb{R}^2\right)\cap\mathcal{N}(U)} \left(z - \mathbb{R}^2_{\geqq}\right).$$

Therefore, we only perform extended Pareto branching when in fact there exits a $\bar{z} \in U$ such that $\bar{z} \leq \underline{z}(\eta)$. This guarantees separation of the branching node.

Note that the EPB requires that all local Nadir points are checked. If the EPB rule is not applied, finding a single local nadir point dominated by the lower bound set leads to the conclusion that branching node η cannot be fathomed. This implies a tradeoff between stronger branching rules and faster treatment of branching nodes that cannot be fathomed.

If we cannot perform either extended Pareto branching or Pareto branching (note that this means, that $x(\eta) \notin \{0,1\}^n$), the branching node is simply separated by a variable dichotomy. The branching variable is chosen based on pseudo–cost information provided by the solver (see for example Achterberg et al. (2005) for a discussion of variable selection strategies in single objective combinatorial optimization).

4. Computational results

In this section we report on the computational experiments conducted with the bi–objective branch–and–cut algorithms for the bi–objective single source capacitated facility location problem (BO–SSCFLP) (see Appendix A). The purpose of the computational study is to answer the following questions

(i) Which implementations based on explicit or implicit lower bound sets perform the best?

(ii) Given an explicit lower bound set, does node fathoming based on linear programming or point–in–polytope algorithms perform the best?

(iii) Is it worth performing extended Pareto branching?

(iv) Is adding cutting planes effective in improving the running time?

Table 2: Different implementations of the bi–objective branch–and–cut algorithm.

Abbreviation[1]	`E-PB-PIP`	`E-PB-LP`	`E-EPB-PIP`	`I-PB-LP`	`I-EPB-LP`
Node selection	Best first	Best first	Best first	Best first	Best first
Cuts	At root node	At root node	At root node	At root note	At root node
Lower bound set	Explicit	Explicit	Explicit	Implicit	Implicit
Fathoming nodes	PIP	LP	PIP	LP	LP
Pareto Branching	Yes	Yes	Yes	Yes	Yes
Extended PB	No	No	Yes	No	Yes

[1] A–B–C. A: Lower bound set (E: explicit, I: implicit). B: Branching strategy (PB: Pareto branching, EPB: extended Preto branching). C: Method for testing the condition in Proposition 1 (LP: linear programming, PIP: point–in–polytope).

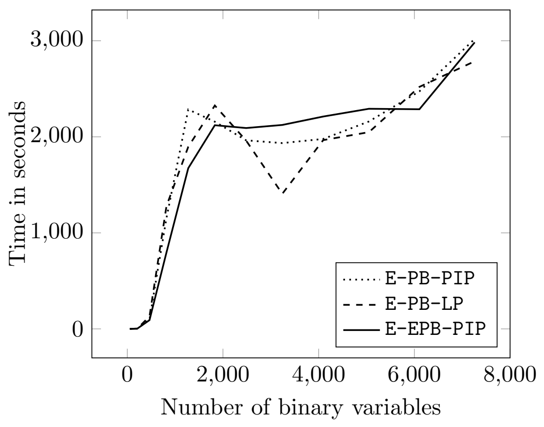
(v) Does the lower bound updating scheme given in Proposition 3 improve the performance?

(vi) Is the bound set based branch–and–cut algorithm competitive with state–of–the art algorithms?

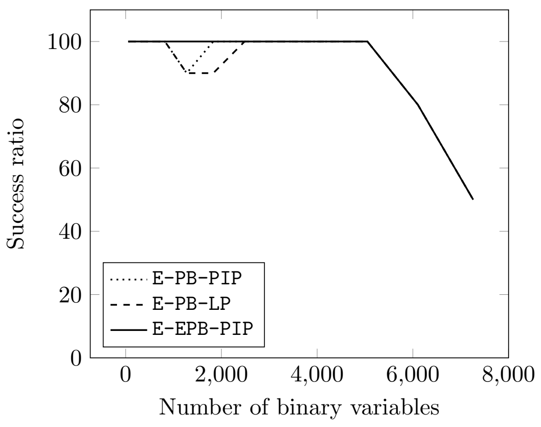
To answer the first three questions above, we have implemented different versions of Algorithm 1. An overview is given in Table 2. The algorithms prefixed with an `E` are all based on explicitly generated lower bound sets while algorithms prefixed with an `I` rely on implicit lower bound sets (see Section 3.4). The branching strategy is indicated using the abbreviations `PB` for Pareto branching and `EPB` for extended Pareto branching (see Section 3.5). For the explicit lower bounds, we proposed two ways of fathoming nodes; one based on linear programming (`LP`) (see Section 3.4.1) and one based on the point–in–polytope (`PIP`) algorithm (see Section 3.4.2). For the algorithms with implicitly given lower bound sets, nodes can only be fathomed using linear programming (see Section 3.4.3). Note that all implementations use a best first search, where the node having the smallest value of $\Lambda^\lambda(\eta)$ is chosen as the next node to be processed. Since computational experience for single objective optimization problems shows that cutting planes have a larger effect at the root note compared to nodes deeper in the tree, we call Algorithm 2 only at the root node. We use general lifted cover inequalities and Fenchel inequalities as cutting planes for the knapsack structures arising from the capacity constraints of the BO–SSCFLP as they have been shown to be effective for the SSCFLP (see e.g. Gadegaard et al. (2015)). Furthermore, Stidsen et al. (2014) established that Pareto branching in bi–objective branch–and–bound gives a significant speed up compared to only branching on variables. We therefore include Pareto branching in all algorithms.

After testing the five implementations specified in Table 2, we address the remaining three questions as follows:

- We answer Question (iv) by comparing the best explicit and implicit implementations with and without cutting planes added.
- We answer Question (v) by comparing the explicit lower bound set based algorithm with and without the updating strategy.
- We finally answer Question (vi) by comparing the overall best implementation of Algorithm 1 with two different implementations of the two–phase method.



(a) Average CPU time in seconds.



(b) Instances solved within the time limit (one hour).

Figure 3: Performance of implementations based on explicit lower bound sets.

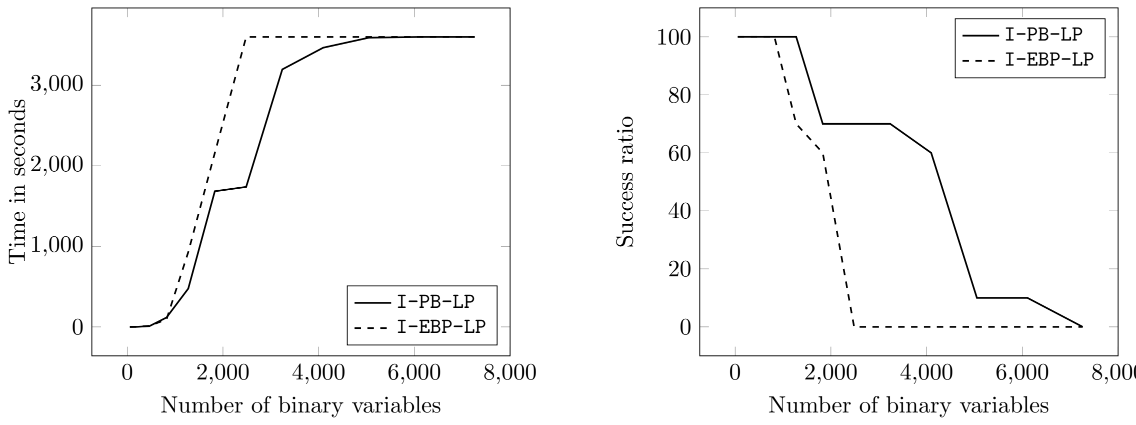
4.1. Implementation details and test instances

All implementations have been coded in C and C++ and compiled using gcc and g++ with optimization option O3 and C++11 enabled. They are all publicly available (see Gadegaard et al. (2016a)). All implementations use CPLEX 12.6 with callbacks as solver. The ParallelMode switch is set to deterministic such that different runs can be compared, the Reduce switch is set such that neither primal nor dual reduction is performed, and all internal cuts of CPLEX are turned off. For all instances a fixed time limit of 3600 CPU seconds (one hour of computation time) is set after which the search is aborted. As CPLEX 12.6 with callbacks is limited to creating at most two child nodes when branching, we only perform extended Pareto branching when there is one or two local nadir points in the set $(L(\eta)+\mathbb{R}^2_{\geqq})$. If there are more local nadir points in $(L(\eta)+\mathbb{R}^2_{\geqq})$, we resort to Pareto branching as explained above. For `E-PB-PIP` and `E-EPB-PIP` the point–in–polytope problem is solved using the `PNPOLY` algorithm developed by Franklin (2006) while all implementations using linear programming for node fathoming are solved by CPLEX using the dual simplex algorithm. Since CPLEX does not allow for changes in the objective function in the callbacks, we use a fixed value of $\lambda = 0.5$ during the branch–and–cut process.

For the computational study we have generated a number of instances of the BO-SSCFLP. These instances were generated in the same way as was done by Stidsen et al. (2014) for the uncapacitated version of the BO–SSCFLP. The demands were generated from the set $\{5,\dots,10\}$ and the capacities from the set $\{10,\dots,20\}$, both according to a uniform distribution. The ratio between the total capacity and total demand is then scaled to equal $r \in \mathbb{R}$, where r is uniformly generated from the interval $[1.5, 4]$. For each instance size, defined by $|\mathcal{I}| \times |\mathcal{J}|$, we have generated 10 instances. The instance generator as well as the instances are all publicly available (see Gadegaard et al. (2016c)). The number of facilities is $|\mathcal{I}| \in \{5, 10, 15, \dots, 60\}$ and the number of customers is set to $|\mathcal{J}| = 2|\mathcal{I}|$. This leads to 120 instances of the SSCFLP ranging in sizes from 5×10 to 60×120, implying the number of binary variables ranges from 55 to 7,260.

4.2. Questions (i)-(iii) - Comparison of implementations

Figure 3 shows a comparison of the implementations based on an explicitly given lower bound set. From Figure 3(a) we see that all implementations based on explicit lower bounds perform



(a) Average CPU time in seconds. (b) Instances solved within the time limit (one hour).

Figure 4: Performance of implementations based on implicit lower bound sets.

equally well. The time in CPU seconds is an average over all instances which could be solved within an hour. All algorithms are able to solve most of the instances with up to 6,000 variables within an hour. However, when the instances grow beyond this size, the success ratio begins to decrease, but as shown in Figure 3(b), 50% of the instances having more than 7,000 variables were still solvable within an hour. Note that `E-PB-PIP` and `E-PB-LP` both fail to solve a single instance having 1,275 variables and `E-PB-LP` also fails to solve one instance having 1,830 binary variables. The algorithm `E-EPB-PIP` does, however, solve all these instances within an hour.

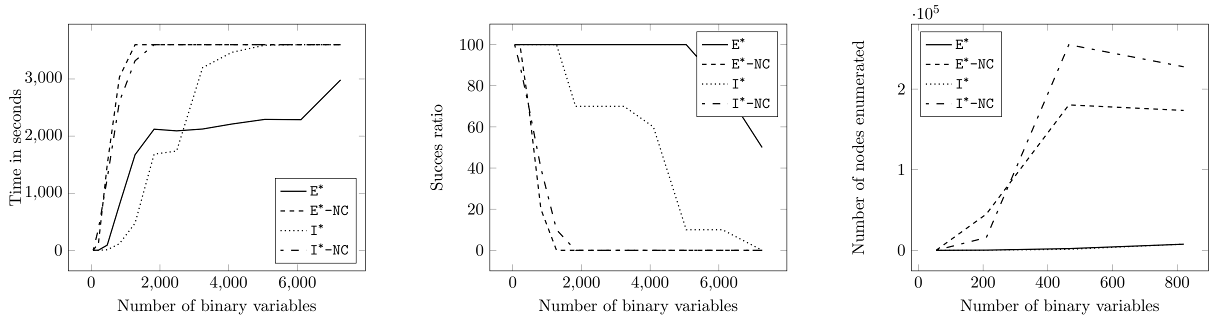
Considering the results obtained for implementations using explicitly given bound sets there is no clear winner, but as `E-EPB-PIP` solves slightly more instances within an hour compared to `E-PB-PIP` and `E-PB-LP`, it therefore seems to be more robust.

Now consider the two implementations based on implicit lower bounds. As is shown in Figure 4(a) the two implementations are comparable for relatively small problem instances, but as the number of binary variables increases, `I-EPB-LP` becomes more time consuming than `I-PB-LP`. From Figure 4(b) we see that `I-EPB-LP` was able to solve instances of up to 1,830 binary variables only, while I–PB–LP performed better by solving instances of up to 5,000 binary variables. The reason is that `I-EPB-LP` has to solve one LP for each local nadir point whereas `I-PB-LP` only has to find one local nadir point for which the linear program (9) has a strictly positive solution value.

When we compare the results against the implementations based on explicit lower bound sets, we see that only significantly smaller instances could be solved using implicit lower bound sets. The computational study of the `E`–implementations showed that in approximately half of the branching nodes the lower bound set is updated using Proposition 3 (see Section 4.4 for a computational study of the effect of Proposition 3), whereas the `I`–implementations need to solve the rather large LPs in (9) several times for each branching node. Especially the `I-EPB-LP` suffers from this problem as more LPs need to be solved in order to perform the extended Pareto branching.

The above tests clearly show, that the best explicit and implicit implementations are `E-EPB-PIP` and `I-PB-LP`.

The overall best implementation seems to be `E-EPB-PIP` which is based on explicit lower bound sets, extended Pareto branching, and a PIP algorithm to solve the fathoming test. It outperforms the best implementation using implicit lower bound sets, the `I-PB-LP`. The use of extended Pareto



(a) Average CPU time in seconds. (b) Instances solved within the time limit (one hour). (c) Number of nodes enumerated.

Figure 5: The effect of cutting planes.

branching seems make the algorithm more robust compared to the other implementations with explicit lower bound sets. Hence in the following, we will only perform further tests with the best implementations under explicit and implicit lower bound sets (`E-EPB-PIP` and `I-PB-LP`). In the remainder of the paper we denote these two implementations `E*` and `I*`, respectively.

4.3. Question (iv) - Is adding cuts worth the effort?

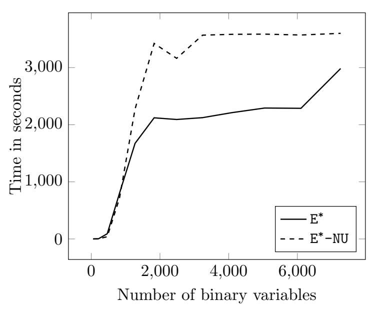
To test whether adding cuts at the root node as explained in Section 3.1 contributes positively to the solution time, we compare the performance of `E*` and `I*` with implementations where the cutting plane algorithm is turned off. We denote these two new variants `E*-NC` and `I*-NC`, where `NC` is an abbreviation for "No Cuts". Instead we let CPLEX generate cutting planes at the root node.

Figure 5 clearly shows that we get a positive effect by adding cuts to the bi-objective LP. The addition of cuts at the root node has a high positive impact on the solution time and solvability of the instances within the one hour limit. It seems that that strong cutting planes are necessary in order to use the bound set branch–and–cut algorithm.

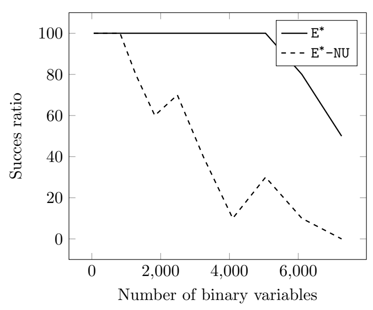
In Figure 5(c) we see that the versions without the initial cutting plane algorithm experience a substantially faster growth in the number of nodes that the algorithms need to enumerate. A peculiar phenomenon is that `I*-NC` enumerates *more* branching nodes than `E*-NC`, even though `E*-NC` updates the lower bound set. This basically means that the lower bound sets used in `E*-NC` are not as strong as those used in `I*-NC`. The explanation seems to be that the solver chooses different search paths for the two algorithms and that the search paths used for `E*-NC` lead to better feasible solutions faster, increasing the fathoming potential. Overall, however, the addition of cuts at the root node has a high impact on the solution times and the instance sizes solvable, and we dare conclude, that cutting planes are necessary in order to solve these instances.

4.4. Question (v) - Effect of using Proposition 3

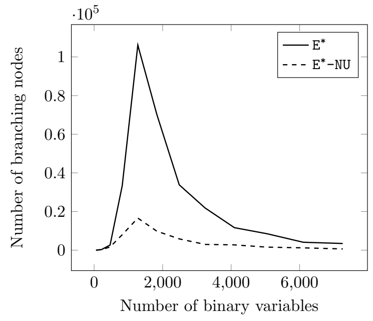
To check if the lower bound updating strategy of Proposition 3 contributes positively to the running times, we modified `E*` such that the bi–objective LP–relaxation is solved in all branching nodes. That is, we obtain the best possible $\mathbb{R}^2_{\geqq}$–convex lower bound set and may fathom nodes faster. This may, however, come at the expense of a higher CPU time needed to solve the bi-objective LP. We name this implementation `E*-NU` where `NU` is an abbreviation for "No Updating".



(a) Average CPU time in seconds.



(b) Instances solved within the time limit (one hour).



(c) Number of nodes enumerated.

Figure 6: Effect of Proposition 3.

In Figure 6(c) it is seen that the E* algorithm produces an order of magnitude more branching nodes compared to the E*-NU. This was expected as the lower bound sets generated by the updating strategy are weaker than those produced by $L^C(\eta)$. Figure 6(a) shows that the updating strategy has a positive effect as the instances grow in size. Furthermore, using the updating strategy makes the procedure more robust as can be seen in Figure 6(b). When we solve a bi–objective LP at each node, some instances become very time consuming, even for smaller sizes. The reason is that solving a degenerate bi–objective LP requires the solution of several degenerate single objective LPs. The updating strategy circumvents this issue by updating the lower bound set via the solution of a *single* objective degenerate LP.

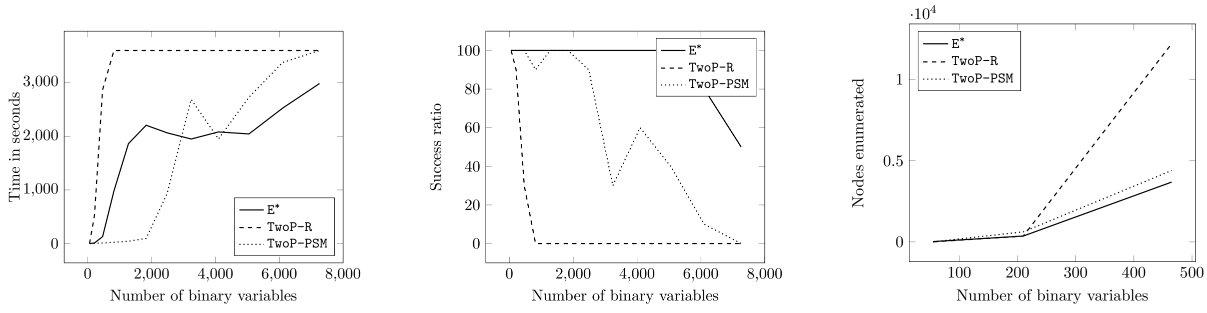
In summary, the updating scheme significantly improves the running time and robustness of the E^* algorithm, and we conclude that the updating scheme is necessary for the branch–and–cut algorithm when solving larger instances.

4.5. Question (vi) - Comparing against the two–phase method

To test the effectiveness of the overall best branch–and–cut approach, we compare E* with two implementations of the two–phase method. Both implementations are publicly available (see Gadegaard et al. (2016b)).

The first two–phase method is implemented as indicated in Stidsen et al. (2014), namely with a second phase based on ranking. We use the abbreviation TwoP-R to denote this two-phase ranking method. The results obtained with the two–phase algorithm seem to be consistent with the results obtained in Stidsen et al. (2014) where facility location problems having up to 20 binary variables could be solved within 300 CPU seconds. Here, we can solve instances of up to 500 binary variables within ten times the computation time. To meet the potential critique that TwoP-R is badly implemented, we here mention that between 99.3% and 100.0% of the running time was spent by CPLEX solving the subproblems. Furthermore, we have implemented TwoP-R such that CPLEX *reoptimizes* the MIPs arising in the subproblems after adding branching constraints. This gives a significant speedup compared to solving each MIP from scratch.

Figure 7 shows the performance of TwoP-R (dashed line) compared to that of E* (solid line). It is clear from Figure 7(a) and Figure 7(b) that the ranking based two–phase method is very inferior to the branch–and–bound algorithm E*. Figure 7(c) gives an explanation of why TwoP-R performs badly compared to E*; When we look at the instances actually solved by TwoP-R, we see a



(a) Average CPU time in seconds.

(b) Instances solved within the time limit (one hour).

(c) Number of nodes enumerated by the algorithms (only instances solved by two–phase method).

Figure 7: Comparison with the two–phase method.

much faster increase in the number of branching nodes to be enumerated in order for the algorithm to solve the instances compared to the `E*`. As `TwoP-R` ranks the solutions between the supported non–dominated solutions found in the first phase, it generates many equivalent solutions leading to a large number of MIPs solved redundantly. Furthermore, even though `TwoP-R` is implemented such that the MIPs are reoptimized after adding no–good constraints, a new root node has to be solved in each MIP leading to excessive computation times.

As the results obtained with this standard ranking based two–phase method were rather disappointing in terms of computation times, we also implemented a two–phase method where the second phase is based on the *perpendicular search method* (PSM) proposed by Chalmet et al. (1986). We denote the two–phase method based on PSM, `TwoP-PSM`. The PSM is basically a branch–and–bound algorithm where an MIP is solved in each node, and where all branching is performed in objective space. The branching strategy in PSM is equivalent to single solution Pareto branching. This implies, that *no* equivalent solutions are generated, and we thereby circumvent this obvious issue with the `TwoP-R`.

Figure 7(a) shows that `TwoP-PSM` (dotted line) performs much better than `TwoP-R` and that it even outperforms `E*` for smaller instances. Although `TwoP-PSM` is faster for the smaller instances, it becomes unstable in the sense that some instances become unsolvable within an hour whereas others can be solved within a few minutes. This can be seen in Figure 7(b) where the success ratio for the `TwoP-PSM` drops much earlier than for the `E*` algorithm. One of the main reasons why the `TwoP-PSM` is faster for small instances is that the preprocessing done by CPLEX reduces the instances considerably. Also, CPLEX finds strong upper bounds very quickly which means that the probing techniques implemented in the solver is able to fix many variables at an early stage of the search. But as the instances grow larger, the preprocessing and the internal heuristics seem to perform worse and more branching nodes need to be enumerated. Both two-phase methods are also slowed down by the scaling of the objective functions performed when searching the triangles created by the first phase; even though the weights used are integers, the difference in the coefficients becomes very large and the instances suffer from *bad scaling* which increases the computation times considerably.

In sum, both two-phase methods in our implementation are outperformed by `E*` on large instances. In particular, the ranking based `TwoP-R` performs very poorly on all instances, whereas the `TwoP-PSM` algorithm has its merits in case of smaller instances.

5. Conclusions

In this paper we have developed a novel bound set based branch–and–cut algorithm for solving bi–objective combinatorial optimization problems. The algorithm was tested using both an explicit and implicit representation of lower bound sets, and we have shown that the best algorithm based on explicit lower bound sets outperforms the best algorithm based on implicit lower bound sets. We proposed an updating scheme that prevented the algorithm from solving a bi–objective LP at each node. Computational results have shown that the cost of weaker lower bound sets was by far outweighed by the improvement in speed when a branching node is processed. The paper also suggests a simple bi–objective cutting plane algorithm that significantly improves the performance of both the explicit and the implicit lower bound based algorithms. Furthermore, we proposed an extension of the Pareto branching strategy suggested in the literature and showed that it makes the explicit lower bound set based algorithm more stable. Finally, we proved the effectiveness of the branch–and–cut algorithm by comparing it to two different implementations of the two–phase method. Especially for larger instances, did the new branch–and–cut algorithm outperform the two–phase methods. An interesting area for future research would be to investigate extensions of the algorithms to multi–objective problems with more than two objectives. Using a generalization of the local nadir points to higher dimensions and the LP based fathoming approaches, both the explicit and implicit lower bound based algorithms could be extended to three or more criteria. Another fruitful area could be to extend the reduction and preprocessing techniques developed for single objective combinatorial optimization to the multi-objective versions. Last, a computational study with problems having a totally unimodular constraint matrix is interesting as feasible solutions can be "harvested" when solving the bi–objective LPs arising in the branching nodes.

6. Acknowledgements

The authors would like to thank Professor Kim Allan Andersen for insightful comments and literature suggestions. This work was partially supported by a grant from Købmand Ferdinand Sallings Mindefond.

Appendices

A. The single source capacitated facility location problem

We have chosen to test our algorithm on the bi–objective single source capacitated facility location problem (BO–SSCFLP) which can be described as follows: given are a set $\mathcal{I}$ of potential facility sites and a set $\mathcal{J}$ of demand points. Each facility has a fixed opening cost of $f_i > 0$ and a capacity $s_i > 0$ while each demand point has a fixed and known demand of $d_j > 0$. Each demand point j must be serviced by exactly one open facility i, resulting in a cost of c_{ij}. It is assumed that all parameters are non–negative integers. The BO–SSCFLP is then the problem of minimizing the total opening cost and the total servicing cost. Given binary variables y_i equaling one only if facility i is open and binary variable x_{ij} equaling one if and only if customer j is serviced by facility

i, the bi–objective SSCFLP can be stated as the BOCO problem

$$\begin{aligned}
\min \quad & \left(\sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}} c_{ij}x_{ij} \ , \ \sum_{i\in\mathcal{I}} f_i y_i \right) \\
\text{s.t.:} \quad & \sum_{i\in\mathcal{I}} x_{ij} = 1, \quad \forall j \in \mathcal{J}, \\
& \sum_{j\in\mathcal{J}} d_j x_{ij} \leq s_i y_i, \quad \forall i \in \mathcal{I}, \\
& x_{ij},\ y_i \in \{0,1\}, \quad \forall j \in \mathcal{J},\ i \in \mathcal{I}.
\end{aligned}$$

The SSCFLP matches the assumption of both objectives having integral values for all feasible solutions. In addition, the bi–objective SSCFLP is a very natural BOCO problem as the objectives are antagonistic by nature: opening an extra facility results in an increase in fixed opening costs and a decrease in servicing costs.

References

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

Aneja, Y., Nair, K., 1979. Bicriteria transportation problem. Management Science 25 (1), 73–78.

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

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

Cohon, J., 1978. Multiobjective Programming and Planning. Academic Presse, INC. (London) LTD.

Dial, R., 1979. A model and algorithm for multicriteria route-mode choice. Transportation Research Part B: Methodological 13 (4), 311–316.

Ehrgott, M., 2005. Multicriteria Optimization, 2nd Edition. Springer berlin, Heidelberg.

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

Fernandez, E., Puerto, J., 2003. Multiobjective solution of the uncapacitated plant location problem. European Journal of Operational Research 145 (3), 509–529.

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

Franklin, W., 2006. PNPOLY - point inclusion in polygon test. Webpage, source code.
URL `http://www.ecse.rpi.edu/~wrf/Research/Short_Notes/pnpoly.html`

Gadegaard, S., Klose, A., Nielsen, L., 2015. An exact algorithm based on cutting planes and local branchin for the single source capacitated facility location problem. Tech. rep., Department of Economics and Business Economics, Aarhus University, working paper.

Gadegaard, S., Nielsen, L., Ehrgott, M., 2016a. A branch and cut algorithm for bi–objective combinatorial optimization problems. GitHub, source code (v0.0.1).
URL `https://github.com/SuneGadegaard/BiObjectiveBranchAndCut`

Gadegaard, S., Nielsen, L., Ehrgott, M., 2016b. A general two–phase method for bi–objective combinatorial optimization. GitHub, source code (v1.0.0).
URL `https://github.com/SuneGadegaard/TwoPhaseMethod`

Gadegaard, S., Nielsen, L., Ehrgott, M., 2016c. An instance generator for the capacitated facility location problem. GitHub, source code (v1.0.0).
URL `https://github.com/SuneGadegaard/SSCFLPgenerator`

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

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

Martello, S., Toth, P., 1990. Knapsack problems: algorithms and computer implementations. John Wiley & Sons, Inc.

Martin, R., 1999. Large scale linear and integer optimization: a unified approach. Kluwer Academic Publishers.

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

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

Nemhauser, G., Wolsey, L., 1988. Integer and Combinatorial Optimization. Wiley New York.

Pedersen, C. R., Nielsen, L., Andersen, K., 2008. The bicriterion multimodal assignment problem: Introduction, analysis, and experimental results. INFORMS Journal on Computing 20 (3), 400–411.

Przybylski, A., Gandibleux, X., Ehrgott, M., 2008. Two phase algorithms for the bi-objective assignment problem. European Journal of Operational Research 185 (2), 509 – 533.

Przybylski, A., Gandibleux, X., Ehrgott, M., 2010. A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives. Discrete Optimization 7 (3), 149 – 165.

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

Schirra, S., 2008. How reliable are practical point-in-polygon strategies? In: Halperin, D., Mehlhorn, K. (Eds.), Algorithms - ESA 2008. Vol. 5193 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 744–755.

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

Sourd, F., Spanjaard, O., Perny, P., 2006. Multi-objective branch and bound. Application to the bi-objective spanning tree problem. In: 7th International Conference in Multi-Objective Programming and Goal Programming.

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

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

Vincent, T., 2009. Multi-objective branch and bound for mixed 0-1 linear programming: Corrections and improvements. Ph.D. thesis, Master's thesis (to appear).

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

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