A Feasibility Pump and Local Search Based Heuristic for Bi-objective Pure Integer Linear Programming

Aritra Pal
Department of Industrial and Management System Engineering, University of South Florida, Tampa, FL, 33620 USA, aritra1@mail.usf.edu

Hadi Charkhgard
Department of Industrial and Management System Engineering, University of South Florida, Tampa, FL, 33620 USA, hcharkhgard@usf.edu, http://www.eng.usf.edu/ hcharkhgard/

We present a new heuristic algorithm to approximately generate the nondominated frontier of bi-objective pure integer linear programs. The proposed algorithm employs a customized version of several existing algorithms in the literature of both single-objective and bi-objective optimization. Our proposed method has two desirable characteristics: (1) There is no parameter to be tuned by users other than the time limit; (2) It can naturally exploit parallelism. An extensive computational study shows the efficacy of the proposed method on some existing standard test instances in which the true frontier is known, and also some large randomly generated instances. We show that even a basic version of our algorithm can significantly outperform NSGA-II (Deb et al. 2002), and the sophisticated version of our algorithm is competitive with MDLS (Tricoire 2012). We also show the value of parallelization on the proposed approach.

Key words: Bi-objective integer linear programming, feasibility pump, local search, the perpendicular search method, parallelization

History:

1. Introduction

Many real-world problems involve multiple objectives. Due to conflict between objectives, finding a feasible solution that simultaneously optimizes all objectives is usually impossible. Consequently, in practice, decision makers want to understand the trade off between objectives before choosing a suitable solution. Thus, generating some of the *efficient solutions*, i.e., solutions in which it is impossible to improve the value of one objective without a deterioration in the value of at least one other objective, is the primary goal in multi-objective optimization.

In recent years, several studies have been conducted on developing exact algorithms for solving multi-objective optimization problems, see for instance Kirlik and Sayın (2014), Köksalan and Lokman (2014), Dächert and Klamroth (2014), Dächert et al. (2012),

Özlen et al. (2013), and Boland et al. (2015b, 2016a,b). Despite this enormous effort, still much of research on multi-objective optimization has been focused on heuristic solution approaches such as evolutionary algorithms. This may be because of the fact that it has been shown that even for the simplest optimization problems, e.g., the minimum spanning tree problem, the shortest path problem, and the matching problem, and even for just two objective functions, determining whether a point in the criterion space is associated with an efficient solution is NP-hard (Papadimitriou and Yannakakis 2000). The EMOO (Evolutionary Multi-objective Optimization) website maintained by Carlos A. Coello Coello (`http://delta.cs.cinvestav.mx/~ccoello/EMOO/`), for example, lists close to 4,700 journal papers and around 3,800 conference papers on the topic. For an introduction to and a recent survey of evolutionary methods for multi-objective optimization see Coello Coello et al. (2007) and Zhou et al. (2011) and for surveys on their use in different domains see Marler and Arora (2004) (engineering), Castillo Tapia and Coello Coello (2007) (economics and finance), Lei (2009) (production scheduling), Ehrgott and Gandibleux (2004) (combinatorial optimization), and Lust and Teghem (2010) (traveling salesman problem). The popularity of evolutionary methods and the importance and relevance of multi-objective optimization is highlighted by the number of citations received by the outstanding paper introducing Nondominated Sorting Genetic Algorithm II (NSGA-II) (Deb et al. 2002): 20,042 according to Google.

The focus of this study is on bi-objective pure integer linear programs. The motivation of our research comes from two observations obtained from the relevant literature. (1) Surprisingly, there is not much effort in developing heuristic solution approaches for (general) bi-objective pure integer linear programs. Two of very few studies in this scope are Tricoire (2012) and Paquete et al. (2007) that present two successful heuristics, the so-called Multi-Directional Local Search (MDLS) and Pareto Local Search (PLS), respectively. In fact, most relevant studies are about developing heuristic algorithms for some specific classes of bi-objective pure integer linear programs. (2) There are many general heuristic algorithms such as NSGA-II that are developed for solving any multi-objective optimization problem. However, these methods are not specifically designed for problems with integer decision variables. These methods have been applied quite successfully for problems with continuous variables but the performance of these methods is not clear on problems with integer decision variables since finding feasible solutions in this case is quite challenging.

This is highlighted by the fact that these methods are usually designed to solve unconstrained multi-objective optimization problems. So, if there are some constraints in the problem, these methods will usually remove them by penalizing them and adding them to the objective functions. So, given that many bi-objective integer linear programs contains multiple constraints, the performance of these algorithms is not clear.

The main contribution of our research is to develop a new heuristic algorithm that is obtained by combining the underlying ideas of several exact/heuristic algorithms in the literature of both single and bi-objective integer linear programs including the perpendicular search method (Chalmet et al. 1986, Boland et al. 2015a), the local search approach (Tricoire 2012), the weighted sum method (Aneja and Nair 1979), and the feasibility pump (Fischetti et al. 2005, Achterberg and Berthold 2007, Boland et al. 2014). The latter has been successfully applied to single-objective integer linear programming, and commercial exact solvers (such as CPLEX) are exploiting the power of this heuristic. However, to the best of our knowledge, we are the first authors proposing a similar technique for bi-objective pure integer linear programs.

In the proposed approach, the feasibility pump is designed for finding feasible solutions by using the weighted sum method. The local search is used for improving the solutions obtained by the feasibility pump heuristic, and the perpendicular search method is used for finding solutions from different parts of the criterion space. Our proposed method has two desirable characteristics. (1) Unlike many heuristic approaches (for instance NSGA-II), there is no parameter to be tuned by users other than the time limit in our proposed method. (2) The proposed approach can naturally exploit parallelism.

We conduct an extensive computational study on a variety of instances. We first show that even the proposed feasibility pump approach can significantly outperform NSGA-II. We then show how the proposed local search approach can improve the results obtained by the feasibility pump heuristic. After that the effect of the perpendicular search method will be explored. We then show that the sophisticated version of our algorithm, containing the feasibility pump, the local search and the perpendicular search methods, is competitive with MDLS. Finally, the effect of parallelization will be shown. To the best of our knowledge, investigating the effect of parallelism is untouched in the literature of solution approaches for multi-objective optimization. So, our research is one of the first studies in this scope.

The remainder of the paper is organized as follows. In Section 2, we introduce notation and some fundamental concepts. In Section 3, we provide the general framework of our algorithm. In Section 4, we introduce the proposed feasibility pump heuristic. In Section 5, we present the proposed local search heuristic. In Section 6, we conduct an extensive computational study. Finally, in Section 7, we give some concluding remarks.

2. Preliminaries

A *multi-objective (pure) integer linear program* (MOILP) can be stated as follows:

$$\min_{x \in \mathcal{X}} \{z_1(x), \ldots, z_p(x)\},$$

where $\mathcal{X} := \left\{x \in \mathbb{Z}^n_+ : Ax \leq b\right\}$ represents the *feasible set in the decision space*, and $z_i(x) := c^i x$ for each $i = 1, \ldots, p$ represents a linear objective function. The image $\mathcal{Y}$ of $\mathcal{X}$ under vector-valued function $z = \{z_1, \ldots, z_p\}$ represents the *feasible set in the objective/criterion space*, i.e., $\mathcal{Y} := z(\mathcal{X}) := \{y \in \mathbb{R}^p : y = z(x) \text{ for some } x \in \mathcal{X}\}$. We denote the linear programming (LP) relaxation of $\mathcal{X}$ by $LP(\mathcal{X})$, and we assume that it is *bounded*. We also assume that all coefficients/parameters are integers, i.e., $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$, and $c^i \in \mathbb{Z}^n$ for each $i = 1, \ldots, p$.

Given that we assume that $LP(\mathcal{X})$ is bounded, for any integer decision variable x_j, there must exist a finite global upper bound u_j. So, it is easy to show that any MOILP can be reformulated as a MOILP with only binary decision variables. For example, we can replace any instance of x_j by $\sum_{k=0}^{\lfloor \log_2 u_j \rfloor} 2^k x'_{j,k}$ where $x'_{j,k} \in \{0,1\}$ in the mathematical formulation. So, in this paper, without loss of generality, we assume that the mathematical formulation contains only binary decision variables, i.e., $\mathcal{X} \subseteq \{0,1\}^n$.

Definition 1. A feasible solution $x^I \in \mathcal{X}$ is called *ideal* if it minimizes all objectives simultaneously, i.e., if $x^I \in \arg\min_{x \in \mathcal{X}} z_i(x)$ for all $i = 1, \ldots, p$.

Definition 2. The point $z^I \in \mathbb{R}^p$ is the *ideal point* if $z^I_i = \min_{x \in \mathcal{X}} z_i(x)$ for all $i \in \{1, \ldots, p\}$.

Note that the ideal point z^I is an imaginary point in the criterion space unless an ideal solution exists. Obviously, if an ideal solution x^I exists then $z^I = z(x^I)$. However an ideal solution often does not exist in practice.

Definition 3. A feasible solution $x' \in \mathcal{X}$ is called *efficient* or *Pareto optimal*, if there is no other $x \in \mathcal{X}$ such that $z_k(x) \leq z_k(x')$ for $k = 1, \ldots, p$ and $z(x) \neq z(x')$. If x' is efficient,

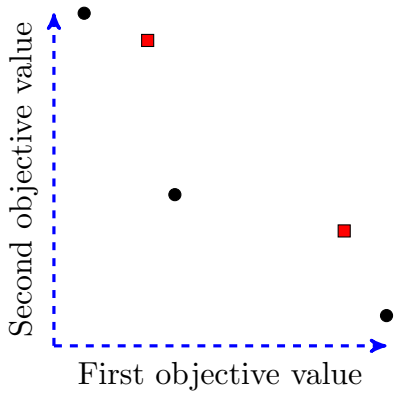


Figure 1 **An illustration of the nondominated frontier of a MOILP with $p = 2$.**

then $z(x')$ is called a *nondominated point*. The set of all efficient solutions $x' \in \mathcal{X}$ is denoted by $\mathcal{X}_E$. The set of all nondominated points $z(x') \in \mathcal{X}$ for some $x' \in \mathcal{X}_E$ is denoted by $\mathcal{Y}_N$ and referred to as the *nondominated frontier* or the *efficient frontier*.

DEFINITION 4. Let $x' \in \mathcal{X}_E$. If there is a $\lambda \in \mathbb{R}^p_>$ such that x' is an optimal solution to $\min_{x \in \mathcal{X}} \lambda^T z(x)$, then x' is called a *supported efficient solution* and $z(x')$ is called a *supported nondominated point*.

Overall, multi-objective optimization is concerned with finding *all* nondominated points, i.e., supported as well as unsupported nondominated points. Specifically, $\min_{x \in \mathcal{X}} z(x)$ is defined to be precisely $\mathcal{Y}_N$. The set of all nondominated points of a MOILP is finite (since by assumption $LP(\mathcal{X})$ is bounded). However, unfortunately, due to the existence of unsupported nondominated points, finding all nondominated points of a MOILP is challenging. An illustration of the nondominated frontier of a MOILP when $p = 2$ is shown in Figure 1 where the (red) rectangles are unsupported.

In this study, we focus on *bi-objective pure integer linear programs*, i.e., MOILPs with $p = 2$, and develop a feasibility pump and local search based heuristic for approximately computing the nondominated frontier of such a problem. Next, we introduce concepts and notation that will facilitate the presentation and discussion of the proposed approach.

We define the points $y^B, y^T \in \mathbb{R}^2$ such that $y^T = (y^T_1, y^T_2) := (-\infty, +\infty)$ and $y^B = (y^B_1, y^B_2) := (+\infty, -\infty)$. In the proposed method, we maintain the list of all $x \in \{0,1\}^n$ that based on which our proposed heuristic was not able to obtain a feasible solution. We call this list as $Tabu$, and in Sections 3 and 4, we explain how the algorithm uses this list. Let $y^1, y^2 \in \mathbb{R}^2$ be two distinct points such that $y^1_1 < y^2_1$ and $y^1_2 > y^2_2$. Note that by our assumptions, all parameters are integers, so we must have that $y^1_1 \leq y^2_1 - 1$ and $y^1_2 \geq y^2_2 + 1$. We

use the operation FeasPump$(Tabu, y^1, y^2)$ to apply a feasibility pump heuristic for computing the (approximate) nondominated frontier of the following bi-objective optimization problem,

$$\min \{z_1(x), z_2(x) : \ x \in \mathcal{X}, \ z_1(x) \leq y_1^2 - 1, \ z_2(x) \leq y_2^1 - 1\}. \tag{1}$$

In Section 4, this operation will be explained in detail. However, in the meantime, we note that since this operation is a heuristic, there is no guarantee about the efficiency of the solutions obtained by this method. So, this operation may just return a set of feasible solutions, denoted by $\mathcal{X}^f \subseteq \mathcal{X}$, with the property that any distinct $x, x' \in \mathcal{X}^f$ do not dominate each other, i.e., there exist $i, j \in \{1, 2\}$ such that $z_i(x) > z_i(x')$ and $z_j(x) < z_j(x')$. Note that for each $x \in \mathcal{X}^f$, $z(x)$ falls into a specific region of the criterion space, i.e.,

$$z(x) \in \{y \in \mathbb{R}^2 : y_1 < y_1^2, y_2 < y_2^1\} \equiv \{y \in \mathbb{R}^2 : y_1 \leq y_1^2 - 1, y_2 \leq y_2^1 - 1\},$$

which is shown by a green box in Figure 2.

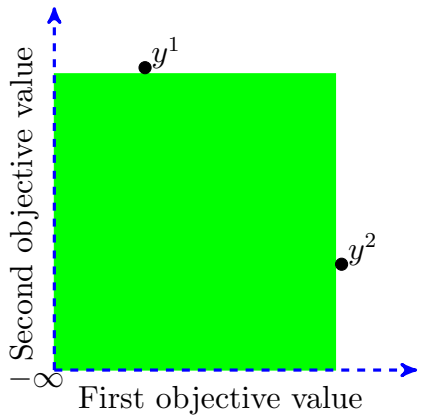


Figure 2 **The search region of FeasPump**(y^1, y^2).

Another operation that we frequently use is a local search heuristic. If the set of feasible solutions returned by the feasibility pump heuristic is not empty, i.e., $\mathcal{X}^f \neq \emptyset$, then we use a local search approach, denoted by LocalSearch$(y^1, y^2, \tilde{x})$, to hopefully improve those feasible solutions, i.e., shifting them toward the true nondominated points, and to possibly find more feasible solutions. In other words, the local search approach seeks to search the same search region defined by the points y^1 and y^2, i.e., the green box in Figure 2, for better (and more) feasible solutions by using $\mathcal{X}^f$. We denote the set of feasible solutions obtained by the local search operation by $\mathcal{X}^l$. This set also has the property that any

distinct $x, x' \in \mathcal{X}^l$ do not dominate each other. In Section 5, this operation will be explained in detail.

In both the local search and feasibility pump operations, we often deal with a set of feasible solutions, $S \subseteq \mathcal{X}$, of which we would like to remove dominated solutions (those that are dominated by other feasible solutions of S). We denote this operation by PARETO(S), and it simply returns the following set,

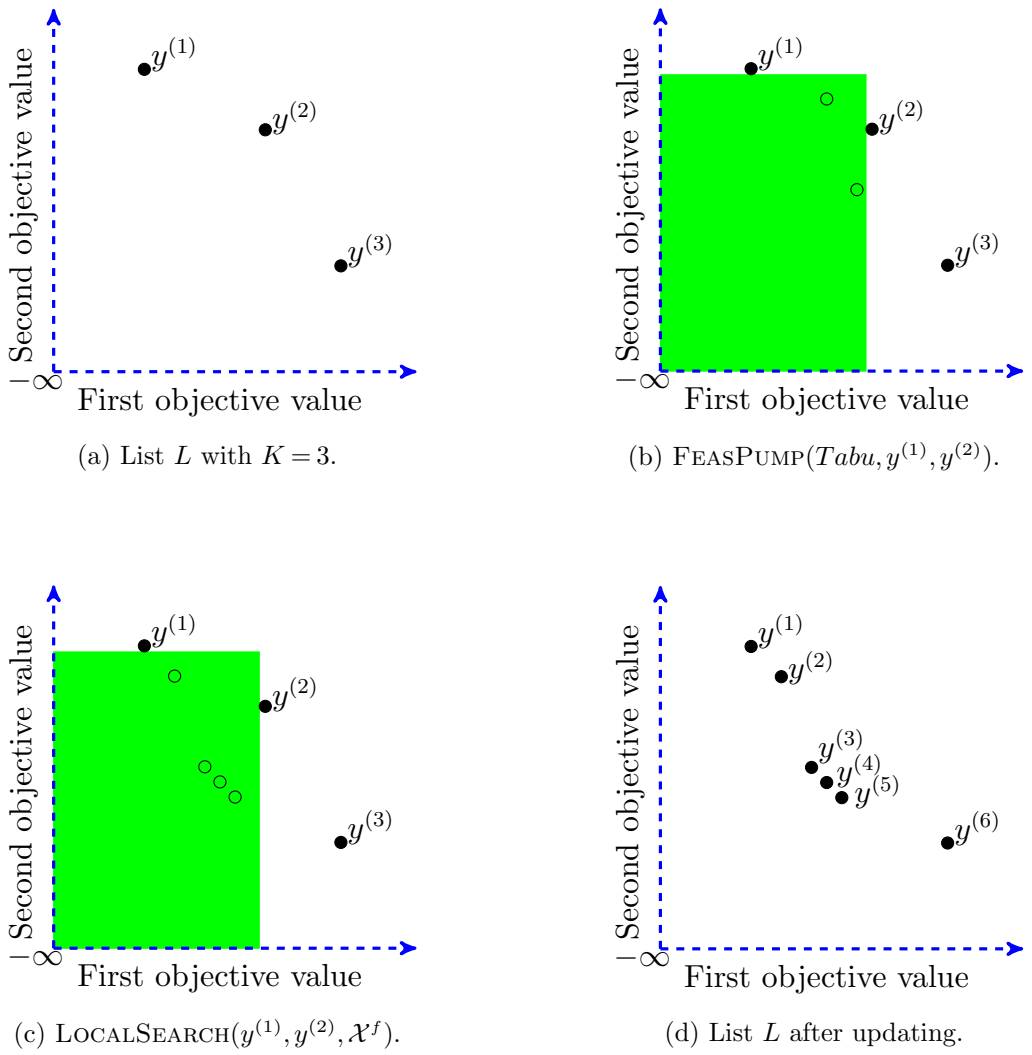
$$\{x \in \mathcal{S} : \nexists x' \in \mathcal{S} \text{ with } z_1(x') \leq z_1(x), z_2(x') \leq z_2(x), z(x') \neq z(x)\}.$$

In this operation, we make sure that the created set is *minimal* in a sense that if there are multiple solutions with the same objective values then only one of them can be in the list.

3. The framework of the proposed algorithm

As we mentioned previously, the algorithm maintains a *Tabu* list which is empty at the beginning. Also, it maintains a set of points $y \in \mathbb{R}^2$, denoted by L, with the property that any two distinct points $y, y' \in L$ do not dominate each other, i.e., there exist $i, j \in \{1, 2\}$ such that $y_i > y'_i$ and $y_j < y'_j$. At the beginning of the algorithm, after creation of L, y^T and y^B will be added to L. However, at the end, both y^T and y^B will be removed from L, and the remaining points will be reported as the approximate nondominated points of the problem. Note that since the proposed approach is a heuristic, there is no guarantee that any point in L is a true nondominated point of the problem.

The algorithm also maintains a queue of pairs of points, denoted by Q, defining search regions in the criterion space. Let $y^{(1)}, \ldots, y^{(K)}$ be all points in L at the beginning of any arbitrary iteration of the algorithm. Obviously, by the definition of L, we can assume that $y_1^{(1)} < \cdots < y_1^{(K)}$ and $y_2^{(1)} > \cdots > y_2^{(K)}$. An illustration of this observation for $K = 3$ is shown in Figure 3a. We construct the queue in such a way that at the beginning of any arbitrary iteration, $(y^{(1)}, y^{(2)}), (y^{(2)}, y^{(3)}), \ldots, (y^{(K-1)}, y^{(K)})$ are the elements of the queue. So, at the beginning of the algorithm, the queue is initialized by the pair (y^T, y^B). It is worth mentioning that the search regions defined in this study are similar to the ones used in the perpendicular search method (Boland et al. 2015a, Chalmet et al. 1986) which is an exact method for solving MOILPs with $p = 2$.



(a) List L with $K = 3$.

(b) FEASPUMP$(Tabu, y^{(1)}, y^{(2)})$.

(c) LOCALSEARCH$(y^{(1)}, y^{(2)}, \mathcal{X}^f)$.

(d) List L after updating.

Figure 3 An illustration of workings of the proposed algorithm.

The algorithm terminates if the solution time reaches the imposed time limit or if the queue is empty, i.e., there is no remaining search region. Next we explain the workings of the algorithm at any arbitrary iteration.

In each iteration, one of the elements of the queue should be popped out. This element is denoted by (y^1, y^2). Note that when an element is popped out then that element does not exist in the queue anymore. Next, the feasibility pump heuristic is used for the search region defined by (y^1, y^2). The result of FEASPUMP(y^1, y^2) is $\mathcal{X}^f$ and the updated $Tabu$ list. An illustration of the points corresponding to solutions in $\mathcal{X}^f$ when $y^1 = y^{(1)}$ and $y^2 = y^{(2)}$ are shown in Figure 3b using unfilled circles. If $\mathcal{X}^f = \emptyset$ then the feasibility pump heuristic has failed to find a feasible solution. So, a new iteration of the algorithm should be started (if possible).

However, if $\mathcal{X}^f \neq \emptyset$ then we apply the local search heuristic to find more (and possibly better) feasible solutions in the search region defined by (y^1, y^2). As is mentioned previously, the set of feasible solutions obtained by LocalSearch$(y^1, y^2, \mathcal{X}^f)$ is denoted by $\mathcal{X}^l$. An illustration of the points corresponding to solutions in $\mathcal{X}^l$ when $y^1 = y^{(1)}$ and $y^2 = y^{(2)}$ is shown in Figure 3c using unfilled circles.

Observe that none of the points in L can dominate the points in the criterion space corresponding to solutions in $\mathcal{X}^l$. So, all elements of $\{z(x) : x \in \mathcal{X}^l\}$ should be added to L. However, some points in L may have been dominated by some elements of $\{z(x) : x \in \mathcal{X}^l\}$. So, the dominated points in L should be removed. Also, the dominated points may still be part of the pairs in the queue for defining search regions. So, those pairs should also be removed from the queue. Consequently, to employ this observation, for each $x \in \mathcal{X}^l$, the following three steps are done in the algorithm:

1. Each element $y \in L$ with $z_1(x) \leq y_1$ and $z_2(x) \leq y_2$ is removed from L. This operation is denoted by $List.clean\big(L, z(x)\big)$.

2. Point $z(x)$ is added to L. This operation is denoted by $List.add\big(L, z(x)\big)$.

3. Each element $(y, y') \in Q$ with $z_1(x) \leq y_1$ and $z_2(x) \leq y_2$ or $z_1(x) \leq y'_1$ and $z_2(x) \leq y'_2$ is removed from Q. This operation is denoted by $Queue.clean\big(Q, z(x)\big)$.

Finally, since all elements of $\mathcal{X}^l$ are new and their corresponding points have been added to L, some new search regions should also be added to the queue. For each $x \in \mathcal{X}^l$, we define $y^{t,x} \in L$ as the closest point to $z(x)$ with $y_1^{t,x} < z_1(x)$ (or equivalently $y_2^{t,x} > z_2(x)$) and $y^{b,x} \in L$ as the closest point to $z(x)$ with $y_1^{b,x} > z_1(x)$ (or equivalently $y_2^{b,x} < z_2(x)$) in L. Consequently, for each $x \in \mathcal{X}^l$, we add $(y^{t,x}, z(x))$ and $(z(x), y^{b,x})$ to the queue. An illustration of the points in L after updating it are shown in Figure 3d.

Algorithm 1 shows a precise description of the proposed feasibility pump and local search based heuristic. We now make two final comments:

- The proposed algorithm can naturally employ parallelism. In other words, each element of the queue Q can be explored in a different processor. We show the value of this feature in Section 6.4.

- To avoid spending too much of valuable computational time on both the feasibility pump and the local search operations in each iteration, we suggest to impose a time limit for the summation of the time spent on these operations in each iteration. We have computationally observed that not more than 12,5% of the total time limit (that is considered

for the entire proposed feasibility pump and the local search based heuristic) should be spent on these operations in each iteration.

Algorithm 1: A feasibility pump and local search based heuristic

```
1  List.create(Tabu)
2  List.create(L)
3  List.add(L, y^T); List.add(L, y^B)
4  Queue.create(Q);
5  Queue.add(Q, (y^T, y^B))
6  while time ≤ Time_Limit & not Queue.empty(Q) do
7      Queue.pop(Q, (y^1, y^2))
8      (X^f, Tabu) ← FeasPump(Tabu, y^1, y^2)
9      if X^f ≠ ∅ then
10         X^l ← LocalSearch(y^1, y^2, X^f)
11         foreach x ∈ X^l do
12             List.clean(L, z(x))
13             List.add(L, z(x))
14             Queue.clean(Q, z(x))
15         foreach x ∈ X^l do
16             Queue.add(Q, (y^{t,x}, z(x)))
17             Queue.add(Q, (z(x), y^{b,x}))
18 List.remove(L, y^T); List.remove(L, y^B)
19 return L
```

4. The feasibility pump

Our proposed feasibility pump approach is similar to the feasibility pump approach used for single-objective integer linear programs. The single-objective version of the feasibility pump works on a pair of solutions $\tilde{x}$ and $\tilde{x}^I$. The first one is feasible for the LP-relaxation of the problem, and so it may be fractional. However, the latter is integer but it is not necessarily feasible for the problem, i.e., geometrically it may lie outside the polyhedron corresponding to the LP-relaxation of the problem. The feasibility pump iteratively updates $\tilde{x}$ and $\tilde{x}^I$, and in each iteration, the hope is to further reduce the distance between $\tilde{x}$ and $\tilde{x}^I$. Consequently, ultimately, we can hope to obtain an integer feasible solution.

In light of the above, we now present our proposed feasibility pump operation. This operation takes the $Tabu$ list, y^1 and y^2 as inputs, and returns the updated $Tabu$ list, and

a set of feasible solutions, $\mathcal{X}^f$. At the beginning, we set $\mathcal{X}^f = \emptyset$. Also, at the end, we call PARETO($\mathcal{X}^f$) to remove the dominated solutions of $\mathcal{X}^f$ before returning it.

Note that by definition, $\mathcal{X}^f$ must be a subset of feasible solutions of Problem (1). To construct $\mathcal{X}^f$, we first try to solve the LP-relaxation of Problem (1), i.e.,

$$\min \{z_1(x), z_2(x) : \ x \in LP(\mathcal{X}),\ z_1(x) \leq y_1^2 - 1,\ z_2(x) \leq y_2^1 - 1\}. \tag{2}$$

This can be done using the well-known *weighted sum method* (WSM) (Aneja and Nair 1979). We denote this operation by WEIGHTEDSUMMETHOD(y^1, y^2), and we explain its details in Section 4.1. In the meantime, we note that this operation returns a subset of efficient solutions for Problem (2), denoted by $\tilde{\mathcal{X}}$. So, these efficient solutions may be fractional.

The next step in the feasibility pump operation is to try to construct an integer feasible solution based on each $\tilde{x} \in \tilde{\mathcal{X}}$ for Problem (1). We set the maximum number of attempts to construct an integer feasible solution based on $\tilde{x} \in \tilde{\mathcal{X}}$ to be equal to the number of variables with fractional values in $\tilde{x}$, denoted by FRACTIONALDEGREE($\tilde{x}$). As an aside, if FRACTIONALDEGREE($\tilde{x}$) = 0 then $\tilde{x}$ is an integer feasible solution, but we set the maximum number of attempts to one (in that case) just to assure that $\tilde{x}$ will be added to $\mathcal{X}^f$ (see Lines 8 to 10 of Algorithm 2). As soon as an integer feasible solution is found, the algorithm will terminate its search for constructing integer feasible solutions based on $\tilde{x}$. Next, we explain the workings of the algorithm during each attempt for constructing an integer feasible solution based on $\tilde{x}$. Note that in the remaining, whenever we say that 'the search will terminate', we mean that 'the search for constructing a feasible solution based on $\tilde{x}$ will terminate'.

During each attempt, we first check whether $\tilde{x}$ is already integer. If it is integer then since $\tilde{x}$ is a feasible solution of Problem (2), we must have that $z_1(\tilde{x}) \leq y_1^2 - 1$ and $z_2(\tilde{x}) \leq y_2^1 - 1$. So, in this case, we add $\tilde{x}$ to $\mathcal{X}^f$, and the search will terminate.

So, in the remaining, we assume that $\tilde{x}$ is not integer. In this case, the algorithm first rounds each component of $\tilde{x}$. This operation is denoted by ROUND($\tilde{x}$), and its result is shown by $\tilde{x}^I$. Now, we first need to check whether $\tilde{x}^I \in \mathcal{X}$ and $z_1(\tilde{x}^I) \leq y_1^2 - 1$ and $z_2(\tilde{x}^I) \leq y_2^1 - 1$. If that is the case then we have found an integer feasible solution. So, again the search will terminate after adding $\tilde{x}^I$ to $\mathcal{X}^f$. So, in the remaining we assume that the search does not terminate after computing $\tilde{x}^I$.

In this case, we first check whether $\tilde{x}^I$ is in the $Tabu$ list. If it is in the $Tabu$ list, we try to modify $\tilde{x}^I$ by using a flipping operation that flips some components of $\tilde{x}^I$, i.e., if a component has the value of zero we may change it to one, and the other way around. We explain the details of this operation in Section 4.2. We denote the flipping operation by FlippingOperation$(Tabu, \tilde{x}, \tilde{x}^I)$, and it returns an updated $\tilde{x}^I$. If $\tilde{x}^I = Null$ then the flipping operation has failed and in this case, the algorithm will terminate its search. However, if $\tilde{x}^I \neq Null$ then there is a possibility that $\tilde{x}^I$ to be an integer feasible solution. So, we again check whether $\tilde{x}^I \in \mathcal{X}$ and $z_1(\tilde{x}^I) \leq y_1^2 - 1$ and $z_2(\tilde{x}^I) \leq y_2^1 - 1$. If that is the case then we have found an integer feasible solution. So, again the search will terminate after adding $\tilde{x}^I$ to $\mathcal{X}^f$.

Finally, regardless of whether $\tilde{x}^I$ is in the $Tabu$ list or not, if the search has not terminated yet, the algorithm takes two further steps:

- It adds $\tilde{x}^I$ to the $Tabu$ list; and
- It updates $\tilde{x}$ using $\tilde{x}^I$ by solving the following optimization problem,

$$\tilde{x} = \arg\min \Big\{ \sum_{j=1:\ \tilde{x}^I_j=0}^{n} x_j + \sum_{j=1:\ \tilde{x}^I_j=1}^{n} (1 - x_j) :\ x \in LP(\mathcal{X}),\ z_1(x) \leq y_1^2 - 1,\ z_2(x) \leq y_2^1 - 1 \Big\},$$

which is denoted by FeasibilitySearch$(y^1, y^2, \tilde{x}^I)$. Basically, the goal of this optimization problem is to find a solution (which is possibly fractional) for Problem (2), which is closer to $\tilde{x}^I$.

Algorithm 2 shows a precise description of the proposed feasibility pump operation.

4.1. The weighted sum method

The WSM operation takes y^1 and y^2 as inputs, and solves the bi-objective LP (2), and returns a set of (possibly) fractional solutions denoted by $\tilde{\mathcal{X}}$. At the beginning of this operation, $\tilde{\mathcal{X}} = \emptyset$.

It is well-known that both the set of efficient solutions and the set of nondominated points of a bi-objective LP are supported and connected, i.e., between any pair of nondominated points there exists a sequence of nondominated points with the property that all points on the line segment between consecutive points in the sequence are also nondominated (Isermann 1977). Consequently, for a bi-objective LP, to describe all nondominated points, it suffices to find all *extreme* supported nondominated points. An illustration of the

Algorithm 2: FEASPUMP$(Tabu, y^1, y^2)$

```
1  X^f ← ∅
2  X̃ ← WEIGHTEDSUMMETHOD(y^1, y^2)
3  foreach x̃ ∈ X̃ do
4      SearchDone ← False
5      t ← 1
6      Max_Iteration ← max{FRACTIONALDEGREE(x̃), 1}
7      while SearchDone = False & t ≤ Max_Iteration do
8          if x̃ ∈ Z^n then
9              X^f ← X^f ∪ {x̃}
10             SearchDone ← True
11         else
12             x̃^I ← ROUND(x̃)
13             if x̃^I ∈ X & z_1(x̃^I) ≤ y_1^2 − 1 & z_2(x̃^I) ≤ y_2^1 − 1 then
14                 X^f ← X^f ∪ {x̃^I}
15                 SearchDone ← True
16             else
17                 if x̃^I ∈ Tabu then
18                     x̃^I ← FLIPPINGOPERATION(Tabu, x̃, x̃^I)
19                     if x̃^I = Null then
20                         SearchDone ← True
21                     else
22                         if x̃^I ∈ X & z_1(x̃^I) ≤ y_1^2 − 1 & z_2(x̃^I) ≤ y_2^1 − 1 then
23                             X^f ← X^f ∪ {x̃^I}
24                             SearchDone ← True
25                 if SearchDone = False then
26                     Tabu.add(x̃^I)
27                     x̃ ← FEASIBILITYSEARCH(y^1, y^2, x̃^I)
28         t ← t + 1
29 X^f ← PARETO(X^f)
30 return (X^f, Tabu)
```

nondominated frontier of a bi-objective LP can be found in Figure 4 in which the extreme supported nondominated points are shown with filled circles.

In light of the above, solving Problem (2) is equivalent to computing its extreme supported nondominated points, and WSM is designed to do so. In WSM, we first find the endpoints of the nondominated frontier. The top endpoint, denoted by $\tilde{y}^T$, can be computed by first solving,

$$\tilde{x}^1 = \arg\min \{z_1(x) : \ x \in LP(\mathcal{X}),\ z_1(x) \leq y_1^2 - 1,\ z_2(x) \leq y_2^1 - 1\},$$

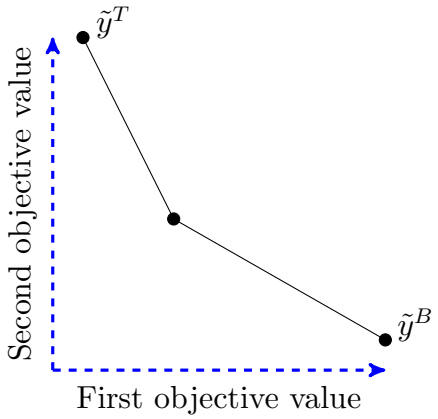
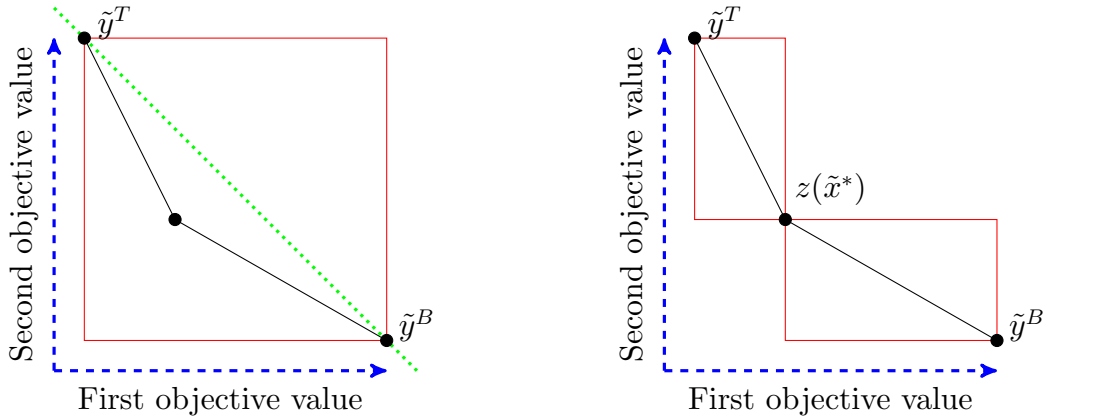


Figure 4 **An illustration of the nondominated frontier of a bi-objective LP.**



(a) $R(\tilde{y}^T, \tilde{y}^B)$ and the new function. (b) The new imaginary rectangles.

Figure 5 **An illustration of the workings of the weighted sum method.**

and if it is feasible, it needs to be followed by solving

$$\tilde{x}^T = \arg\min \{z_2(x) : \ z_1(x) \leq z_1(\tilde{x}^1), \ x \in LP(\mathcal{X}), \ z_1(x) \leq y_1^2 - 1, \ z_2(x) \leq y_2^1 - 1\}.$$

Obviously, $\tilde{y}^T = (z_1(\tilde{x}^T), z_2(\tilde{x}^T))$. We denote the operation of computing $\tilde{x}^T$ by

$$\arg\operatorname{lex}\min\{z_1(x), z_2(x) : x \in LP(\mathcal{X}), \ z_1(x) \leq y_1^2 - 1, \ z_2(x) \leq y_2^1 - 1\}.$$

Similarly, the bottom endpoint, denoted by $\tilde{y}^B$, can be computed by first solving,

$$\tilde{x}^2 = \arg\min \{z_2(x) : \ x \in LP(\mathcal{X}), \ z_1(x) \leq y_1^2 - 1, \ z_2(x) \leq y_2^1 - 1\},$$

and if it is feasible, it needs to be followed by solving

$$\tilde{x}^B = \arg\min \{z_1(x) : \ z_2(x) \leq z_2(\tilde{x}^2), \ x \in LP(\mathcal{X}), \ z_1(x) \leq y_1^2 - 1, \ z_2(x) \leq y_2^1 - 1\}.$$

Obviously, $\tilde{y}^B = (z_1(\tilde{x}^B), z_2(\tilde{x}^B))$. We denote the operation of computing $\tilde{x}^B$ by

$$\arg\operatorname{lex\,min}\{z_2(x), z_1(x) : x \in LP(\mathcal{X}),\ z_1(x) \leq y_1^2 - 1,\ z_2(x) \leq y_2^1 - 1\}.$$

An illustration of the endpoints can be found in Figure 4. It is evident that if $\tilde{x}^T = null$, i.e., the bi-objective LP (2) is infeasible, then we must have that $\tilde{x}^B = null$. So, we do not need to compute it. Also, if $\tilde{y}^T = \tilde{y}^B$ then the bi-objective LP (2) has an ideal point. So, in the remaining we assume that $\tilde{y}^T \neq \tilde{y}^B$.

After computing the endpoints of the nondominated frontier, we add their corresponding solutions to $\tilde{\mathcal{X}}$ (if they exist). It is evident that all other extreme supported nondominated points must be in the imaginary rectangle defined by $\tilde{y}^T$ and $\tilde{y}^B$, denoted by $R(\tilde{y}^T, \tilde{y}^B)$ (See Figure 5a). The weighted sum method then explores this rectangle and may change it into smaller rectangles, and repeat this process in each until it finds all extreme supported nondominated points. More precisely, to explore a rectangle $R(\tilde{y}^1, \tilde{y}^2)$ with $\tilde{y}_1^1 < \tilde{y}_1^2$ and $\tilde{y}_2^1 > \tilde{y}_2^2$, the following weighted sum optimization problem should be solved,

$$\tilde{x}^* = \arg\min\ \{\lambda_1 z_1(x) + \lambda_2 z_2(x) :\ x \in LP(\mathcal{X}),\ z_1(x) \leq y_1^2 - 1,\ z_2(x) \leq y_2^1 - 1\},$$

with $\lambda_1 = \tilde{y}_2^1 - \tilde{y}_2^2$ and $\lambda_2 = \tilde{y}_1^2 - \tilde{y}_1^1$, i.e., the objective function is parallel to the line that connects $\tilde{y}^1$ and $\tilde{y}^2$ in the criterion space. Figure 5a shows an example with $\tilde{y}^1 = \tilde{y}^T$ and $\tilde{y}^2 = \tilde{y}^B$.

It is easy to see that the optimum point $z(\tilde{x}^*)$ is an as yet unknown supported nondominated point if $\lambda_1 z_1(\tilde{x}^*) + \lambda_2 z_2(\tilde{x}^*) < \lambda_1 \tilde{y}_1^1 + \lambda_2 \tilde{y}_2^1$. That is, the optimization either returns a new supported nondominated point $z(\tilde{x}^*)$ or a convex combination of $\tilde{y}^1$ and $\tilde{y}^2$. When an as yet unknown supported nondominated point $\tilde{y}^*$ is returned, the method is applied recursively to search $R(\tilde{y}^1, z(\tilde{x}^*))$ and $R(z(\tilde{x}^*), \tilde{y}^2)$ for additional as yet unknown locally supported nondominated points (see Figure 5b). Note that when an as yet unknown supported nondominated point is returned, it is not necessarily an extreme supported nondominated point. However, after termination of WSM, the set of all extreme supported nondominated points is a subset of the generated points.

Algorithm 3 shows a precise description of WSM. Overall, we have computationally observed that the performance of our proposed feasibility pump and local search based heuristic improves, if we terminate the WSM as soon as the cardinality of $\tilde{\mathcal{X}}$ does not satisfy the following condition:

$$|\tilde{\mathcal{X}}| \leq \min\{5\lceil \log_2 h \rceil, 50\},$$

where $h = \max\{m, n\}$. Note that based on our discussion in Section 2, m is the number of constraints and n is the number of variables.

Algorithm 3: WEIGHTEDSUMMETHOD(y^1, y^2)

$Queue.create(\tilde{Q})$;
$\tilde{\mathcal{X}} \leftarrow \emptyset$
$\tilde{x}^T \leftarrow \arg\operatorname{lex\,min}\{z_1(x), z_2(x) : x \in LP(\mathcal{X}),\ z_1(x) \le y_1^2 - 1,\ z_2(x) \le y_2^1 - 1\}$
if $\tilde{x}^T \neq Null$ **then**
    $\tilde{\mathcal{X}} \leftarrow \tilde{\mathcal{X}} \cup \{\tilde{x}^T\}$
    $\tilde{x}^B \leftarrow \arg\operatorname{lex\,min}\{z_2(x), z_1(x) : x \in LP(\mathcal{X}),\ z_1(x) \le y_1^2 - 1,\ z_2(x) \le y_2^1 - 1\}$
    if $z(\tilde{x}^T) \neq z(\tilde{x}^B)$ **then**
        $\tilde{\mathcal{X}} \leftarrow \tilde{\mathcal{X}} \cup \{\tilde{x}^B\}$
        $\tilde{Q}.add\big(R(z(\tilde{x}^T), z(\tilde{x}^B))\big)$
while *not* $Queue.empty(\tilde{Q})$ **do**
    $Queue.pop(\tilde{Q}, R(\tilde{y}^1, \tilde{y}^2))$
    $\lambda_1 \leftarrow \tilde{y}_2^1 - \tilde{y}_2^2$
    $\lambda_2 \leftarrow \tilde{y}_1^2 - \tilde{y}_1^1$
    $\tilde{x}^* \leftarrow \arg\min\ \{\lambda_1 z_1(x) + \lambda_2 z_2(x) :\ x \in LP(\mathcal{X}),\ z_1(x) \le y_1^2 - 1,\ z_2(x) \le y_2^1 - 1\}$
    if $\lambda_1 \tilde{y}_1^* + \lambda_2 \tilde{y}_2^* < \lambda_1 \tilde{y}_1^1 + \lambda_2 \tilde{y}_2^1$ **then**
        $\tilde{\mathcal{X}} \leftarrow \tilde{\mathcal{X}} \cup \{\tilde{x}^*\}$
        $\tilde{Q}.add\big(R(\tilde{y}^1, z(\tilde{x}^*))\big)$
        $\tilde{Q}.add\big(R(z(\tilde{x}^*), \tilde{y}^2)\big)$
return $\tilde{\mathcal{X}}$

4.2. Flipping operation

The flipping operation takes the $Tabu$ list, the fractional solution $\tilde{x}$, and the integer solution $\tilde{x}^I$ as inputs. Obviously, by construction, $\tilde{x}^I$ is in the Tabu list. So, this operation tries to flip the components of $\tilde{x}^I$, to construct an integer solution that it is not in the $Tabu$ list. Note that such a solution is not necessarily feasible for Problem (1). As soon as one is found, the flipping operation terminates, and the result, denoted by x^I, will be returned. At the beginning, we set $x^I = Null$.

The flipping operation contains two parts including a deterministic procedure and a stochastic procedure. The stochastic procedure will be activated only if the deterministic fails to find an integer solution, i.e., if x^I is still null after the deterministic procedure. It is

worth mentioning that we have computationally observed that this order performs better than first doing the stochastic procedure and then deterministic procedure.

Before explaining these two procedures, we introduce a new operation denoted by SortIndex$(\tilde{x}, \tilde{x}^I)$. This operation simply sorts the set $\{j \in \{1,\ldots,n\} : |\tilde{x}_j - \tilde{x}^I_j| \neq 0\}$ based on the value of $|\tilde{x}_j - \tilde{x}^I_j|$ from large to small. Note that, by construction, if $\tilde{x}_j$ is not fractional then $\tilde{x}^I_j = \tilde{x}_j$ (because $\tilde{x}^I_j$ is the rounded value of $\tilde{x}_j$). This implies that this operation sorts the index set of fractional components of $\tilde{x}$. We denote the result of this operation by $\{e^1,\ldots,e^M\}$ where

$$|\tilde{x}_{e^k} - \tilde{x}^I_{e^k}| \geq |\tilde{x}_{e^{k+1}} - \tilde{x}^I_{e^{k+1}}|$$

for each $k = 1,\ldots,M-1$.

The deterministic procedure attempts at most M times to construct an integer solution. At beginning we make a copy of $\tilde{x}^I$, and denote it by $\hat{x}^I$. During attempt $j \leq M$, we flip the component $\hat{x}^I_{e^j}$. This implies that if $\hat{x}^I_{e^j} = 1$ then we make it equal to zero, and the other way around. We denote this operation by Flip$(\hat{x}^I, e^j)$. If the result is not in the $Tabu$ list then the algorithm terminates, and we report $\hat{x}^I$. Otherwise, (if possible) a new attempt will be started.

The stochastic procedure also attempts at most M times to construct an integer solution. At each attempt, we make a copy of $\tilde{x}^I$, and denote it by $\hat{x}^I$, we then randomly select an integer number from the interval $[\lceil \frac{M}{2} \rceil, M-1]$. This operation is denoted by RandBetween$(\lceil \frac{M}{2} \rceil, M-1)$, and its result is denoted by Num. We then randomly generate a set denoted by R such that $R \subseteq \{1,\ldots,M\}$ and $|R| = Num$. The operation is denoted by RandomlyChoose$(Num, \{1,\ldots,M\})$. We then flip the component $\hat{x}^I_r$ for each $r \in R$. If the result is not in the $Tabu$ list then the algorithm terminates, and we report $\hat{x}^I$. Otherwise, (if possible) a new attempt will be started.

Algorithm 4 shows a precise description of FlippingOperation$(Tabu, \tilde{x}, \tilde{x}^I)$.

5. The local search

The local search operation (that we have developed) is basically a 1-Opt local search algorithm (Lin and Kernighan 1973) that takes y^1, y^2, and the result of the feasibility pump operation $\mathcal{X}^f$ as inputs. As mentioned previously, this operation returns a set of feasible solutions for Problem (1), denoted by $\mathcal{X}^l$, using $\mathcal{X}^f$. At the beginning of the algorithm, we set $\mathcal{X}^l = \mathcal{X}^f$. At the end, we call Pareto$(\mathcal{X}^l)$ to remove the dominated solutions of $\mathcal{X}^l$ before returning it.

Algorithm 4: FLIPPINGOPERATION($Tabu, \tilde{x}, \tilde{x}^I$)

$\{e^1, \ldots, e^M\} \leftarrow$ SORTINDEX($\tilde{x}, \tilde{x}^I$)
$x^I \leftarrow Null$
$\hat{x}^I \leftarrow \tilde{x}^I$
$j \leftarrow 1$
while $j \leq M$ *and* $x^I = Null$ **do**
    $\hat{x}^I \leftarrow$ FLIP($\hat{x}^I, e^j$)
    if $\hat{x}^I \notin Tabu$ **then**
        $x^I \leftarrow \hat{x}^I$
    else
        $j \leftarrow j + 1$
$j \leftarrow 1$
while $j \leq M$ *and* $x^I = Null$ **do**
    $\hat{x}^I \leftarrow \tilde{x}^I$
    $Num \leftarrow$ RANDBETWEEN($\lceil \frac{M}{2} \rceil, M - 1$)
    $R \leftarrow$ RANDOMLYCHOOSE($Num, \{1, \ldots, M\}$)
    foreach $r \in R$ **do**
        $\hat{x}^I \leftarrow$ FLIP($\hat{x}^I, r$)
    if $\hat{x}^I \notin Tabu$ **then**
        $x^I \leftarrow \hat{x}^I$
    else
        $j \leftarrow j + 1$
return x^I

For each $x^f \in \mathcal{X}^f$, the algorithm tries to generate at most n new (integer) feasible solutions for Problem (1), and add them to $\mathcal{X}^l$. More precisely, for each $x^f \in \mathcal{X}^f$ and each $j \in \{1, \ldots, n\}$, the algorithm first makes a copy of x^f, denoted by x^{new}. We then explore the consequence of flipping component x_j^{new}. Obviously, if $x_j^{new} = 1$ and either $c_j^1 > 0$ or $c_j^1 > 0$ or both, then the flipping can improve the value of at least one objective function. So, in this case, we check whether the new solution after flipping is feasible for Problem (1), and if that is the case then we add it to $\mathcal{X}^l$. Similarly, if $x_j^{new} = 0$ and either $c_j^1 < 0$ or $c_j^1 < 0$ or both, then the flipping can improve the value of at least one objective function. So, again, in this case, we check whether the new solution after flipping is feasible for Problem (1), and if that is the case then we add it to $\mathcal{X}^l$.

Algorithm 5 shows a precise description of LOCALSEARCH($y^1, y^2, \mathcal{X}^f$). We now make a comment. Checking whether x^{new} after flipping is feasible for Problem (1) can be done efficiently. Without loss of generality suppose that all constraints are either in the form of

$\leq$ or $=$. We can compute the *slack*, i.e., the difference between right hand side value and left hand side value, of each constraint of Problem (1) for x^{new} before flipping. Obviously for the constraints of the equality form the slack is always zero. We then sort the slacks in non-decreasing order, and check them one by one to see whether any of them will be violated by flipping the component x_j^{new}. If none of the slacks are violated then the solution after flipping must be feasible for Problem (2). For example, suppose that before flipping, we have that $x_j^{new} = 0$. Also suppose that the slack of an arbitrary constraint is s. If the coefficient of x_j^{new} in that constraint is greater than s then it implies that flipping results in a solution which violates that particular constraint. So, x_j^{new} after flipping cannot be a feasible solution for Problem (2). Finally, we note that if there are too many equality constraints in Problem (2), the local search operation most likely will fail to improve the feasible solutions obtained by the feasibility pump. So, we only activate this operation if at least 5% of constraints are inequalities.

Algorithm 5: LocalSearch$(y^1, y^2, \mathcal{X}^f)$

```
1  X^l ← X^f
2  foreach x^f ∈ X^f do
3      foreach j ∈ {1,...,n} do
4          x^new ← x^f
5          if x_j^f = 1 and (c_j^1 > 0 or c_j^2 > 0) then
6              x_j^new ← 0
7              if x^new ∈ X and z_1(x^new) ≤ y_1^2 − 1 and z_2(x^new) ≤ y_2^1 − 1 then
8                  X^l ← X^l ∪ {x^new}
9          else
10             if x_j^f = 0 and (c_j^1 < 0 or c_j^2 < 0) then
11                 x_j^new ← 1
12                 if x^new ∈ X and z_1(x^new) ≤ y_1^2 − 1 and z_2(x^new) ≤ y_2^1 − 1 then
13                     X^l ← X^l ∪ {x^new}
14 X^l ← Pareto(X^l)
15 return (X^f, Tabu)
```

6. Computational results

To evaluate the performance of the proposed method, we conduct a comprehensive computational study. We use the Julia programming language, and the package MathProgBase.jl (Lubin and Dunning 2015) to implement the proposed approach, and employ CPLEX

12.7 as the single-objective linear programming solver. All computational experiments are carried out on a Dell PowerEdge R630 with two Intel Xeon E5-2650 2.2 GHz 12-Core Processors (30MB), 128GB RAM, and the RedHat Enterprise Linux 6.8 operating system. We only use a single thread for all experiments except those about parallelization (Section 6.4). The code and instances used in this study can all be found at `https://goo.gl/fwgHjO` and `https://goo.gl/mevqbk`, respectively. It is worth mentioning that our code is generic and user-friendly in a sense that users only have to provide an input file with (a specific) lp-format to test our code on any BOILP.

Table 1 The list of existing instances in the relevant literature used in this study.

Problem	#Constraints	#Variables	#Instances	Source(s)
Knapsack problem	1	50	29	Gandibleux and Freville (2000), Degoutin and Gandibleux (2002), Captivo et al. (2003),Soylu (2015)
		100	7	
		150	6	
		200	5	
		250		
		300		
		350		
		400		
		450		
		500		
		600	1	
		700		
		800		
		900		
		1,000		
	2	375	5	Boland et al. (2015a)
		500		
		625		
		750		
Assignment problem	400	40,000	10	
	600	90,000		
Set covering problem	10	100	4	Soylu (2015)
	40	200		
	60	600		
Set packing problem	300	100	24	Delorme et al. (2010)
	500			
	600	200	36	
	1,000			

We found a total of 254 instances corresponding to different bi-objective integer linear programs in the relevant literature that are used frequently in multiple studies. Some necessary information about these instances such as the size and some studies that have used them can be found in Table 1. It is worth mentioning that many of these instances take several hours to be solved using exact solution approaches. Also, most of our experiments are conducted on these 254 instances. However, we randomly generate 60 large instances for conducting our parallelization experiments that we will discuss in Section 6.4.

To show the performance of our proposed algorithm, and the feasibility pump and local search operations, we use different versions of the proposed algorithm in this computational study:

- **V1:** This version is designed for showing the value of the proposed feasibility pump operation. So, the underlying question/idea is that if we want to run just the feasibility pump operation once how good would be the solutions obtained by this operation? So, we define V1 precisely as FeasPump$(\emptyset, y^T, y^B)$;
- **V2:** This version is designed for showing the value of the local search operation. So, again, the underlying question/idea is that if we want to run just the local search operation once how good would be the solutions obtained by this operation? However, to be able to run the local search operation once, we need to first run the feasibility pump operation once. So, we define V2 precisely as FeasPump$(\emptyset, y^T, y^B)$ + LocalSearch$(y^T, y^B, \mathcal{X}^0)$, where $\mathcal{X}^0$ is the solutions obtained by FeasPump$(\emptyset, y^T, y^B)$; and
- **V3:** We define V3 precisely as our proposed feasibility pump and local search based algorithm, i.e., Algorithm 1.

Note that in this computational study, for all 254 instances, we know the true nondominated frontier, i.e., $\mathcal{Y}_N$. Also, the algorithms that we employ are all heuristics, and so they just generate an approximate nondominated frontier, denoted by $\mathcal{Y}_N^A$. However, since $\mathcal{Y}_N^A$ is not necessarily equal to $\mathcal{Y}_N$ (in fact $\mathcal{Y}_N^A$ may not be even a subset of $\mathcal{Y}_N$), evaluating the quality of $\mathcal{Y}_N^A$ is crucial (Sayın 2000). In order to do so, it is common to normalize the points in $\mathcal{Y}_N$ and $\mathcal{Y}_N^A$. We assume that an arbitrary point $y \in \mathbb{R}^2$ in the criterion space should be normalized as follows:

$$\left(\frac{y_1 - \min_{\bar{y} \in \mathcal{Y}_N} \bar{y}_1}{\max_{\bar{y} \in \mathcal{Y}_N} \bar{y}_1 - \min_{\bar{y} \in \mathcal{Y}_N} \bar{y}_1}, \frac{y_2 - \min_{\bar{y} \in \mathcal{Y}_N} \bar{y}_2}{\max_{\bar{y} \in \mathcal{Y}_N} \bar{y}_2 - \min_{\bar{y} \in \mathcal{Y}_N} \bar{y}_2}\right).$$

Obviously this guarantees that if $y \in \mathcal{Y}_N$ then the components of its corresponding normalized point take on values from the interval $[0, 1]$. So, in the remaining, we assume that all points in $\mathcal{Y}_N$ and $\mathcal{Y}_N^A$ are already normalized. We next review a few techniques that can be used to evaluate the quality of an approximate nondominated frontier when the true nondominated frontier is known (for further details please refer to Boland et al. (2016a)). However, before that, we have to first introduce some new notation. We denote the Euclidean distance between two points y and y' by $d(y, y')$. Define $k(y)$ to be the closest point in the approximate frontier to a true nondominated point $y \in \mathcal{Y}_N$. Finally, for each

$y \in \mathcal{Y}_N^A$, define $n(y)$ to be the number of (true) nondominated points $y' \in \mathcal{Y}_N \setminus \mathcal{Y}_N^A$ with $k(y') = y$.

• **Hypervolume gap.** One of the best-known measures for assessing and comparing approximate frontiers is the *hypervolume indicator* (or S-metric), which is the volume of the dominated part of the criterion space with respect to a reference point (Zitzler et al. 2007, 2003). As the reference point impacts the value of the hypervolume indicator, we use the *nadir point*, i.e., $z_i^N := \max\{y_i : y \in \mathcal{Y}_N\}$ for $i = 1, 2$, as the reference point, which is the best possible choice for the reference point. Note that computing the nadir point is not easy in general, but trivial once the nondominated frontier is known. Given that the entire nondominated frontier is known for all our instances, we define the hypervolume gap as follows,

$$\frac{100 \times (\text{ Hypervolume of } \mathcal{Y}_N - \text{ Hypervolume of } \mathcal{Y}_N^A)}{\text{Hypervolume of } \mathcal{Y}_N}.$$

As a consequence, an approximate nondominated frontier with smaller hypervolume gap is more desirable.

• **Cardinality.** We define the cardinality of an approximate frontier simply as the percentage of the points of the true nondominated points it contains, i.e., $\frac{100|\mathcal{Y}_N^A \cap \mathcal{Y}_N|}{|\mathcal{Y}_N|}$. As a consequence, an approximate nondominated frontier with larger cardinality is more desirable.

• **Coverage.** A simple coverage indicator can be defined as

$$f^a := \frac{\sum_{y \in \mathcal{Y}_N \setminus \mathcal{Y}_N^A} d(k(y), y)}{|\mathcal{Y}_N \setminus \mathcal{Y}_N^A|},$$

i.e., the average distance to the closest point in the approximate frontier. Observe that f^a can be viewed as a measure of dispersion of the points in the nondominated frontier. Smaller values of f^a indicate that the nondominated points in the approximate frontier are in different parts of the criterion space. As a consequence, an approximate nondominated frontier with smaller f^a is more desirable.

• **Uniformity.** A uniformity indicator should capture how well the points in an approximate frontier are spread out. Points in a cluster do not increase the quality of an approximate frontier. We define the uniformity indicator to be

$$\mu := \frac{\sum_{y \in \mathcal{Y}_N^A} n(y)}{|\mathcal{Y}_N^A|}.$$

As a consequence, an approximate nondominated frontier with smaller μ is more desirable.

6.1. V1 vs NSGA-II

In this section, we compare V1 with NSGA-II. We again note that one of the features of V1 (and in general our proposed feasibility pump and local search based algorithm) is that there is no parameter (other than the time limit) that needs to be defined by users. However, since NSGA-II is a Genetic Algorithm, it has several parameters that need to be defined by users. Two of the most important ones are the *population size* (denoted by Pop) and the *number of generations* (denoted by g). We construct a total of 4 different settings for these two parameters. Let $\alpha, \beta \in \mathbb{R}_+$ be two user-defined parameters. To construct the four different settings, we assume that $\alpha, \beta \in \{2, 4\}$.

- In NSGA-II, Pop must be a multiplier of four. So, we set Pop to the minimum multiplier of four such that $Pop \geq \alpha|\mathcal{Y}_N|$.
- We set $g = \beta|\mathcal{Y}_N|$.

It is also worth mentioning that in NSGA-II, we also set the value of parameters entitled *the probability of crossover* and *the probability of mutation* to 0.9 and 0.5, respectively. We impose a run time limit of 60 seconds for each setting. We then combine the (integer) feasible solutions obtained in each setting, and compare the accumulated results of NSGA-II with the results obtained by V1. Note that we do not impose any time limit for V1 since it naturally terminates in a fraction of a second for almost all instances (for some of them, it takes up to 10 seconds).

From the total of 254 instances, NSGA-II is able to find at least one (integer) feasible solution for only 40 (small) instances. Also, it is not able to find any feasible solution for any of the instances of (bi-objective) assignment and set packing problems. On the other hand, V1 is able to find some feasible solutions for all 254 instances. Next, we compare NSGA-II and V1 only on those 40 instances, and make two comments.

- The number of points in the approximate frontier generated by V1 is significantly larger than the number of points in the approximate frontier generated by NSGA-II. This can be easily observed from Figure 6.
- The approximate nondominated frontier generated by NSGA-II is completely dominated by the approximate nondominated frontier obtained by V1 for all instances. This implies that for any point in the approximate nondominated frontier generated by NSGA-II, there is at least one point in the approximate nondominated frontier generated by V1 that is better, i.e., dominates it. Figure 7 shows the minimum Euclidean distance between

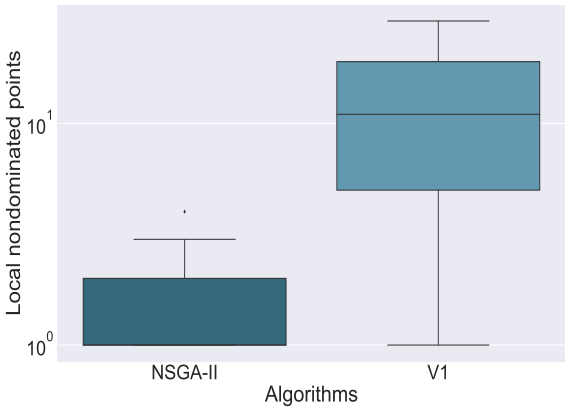


Figure 6 **The number of points in the approximate frontiers.**

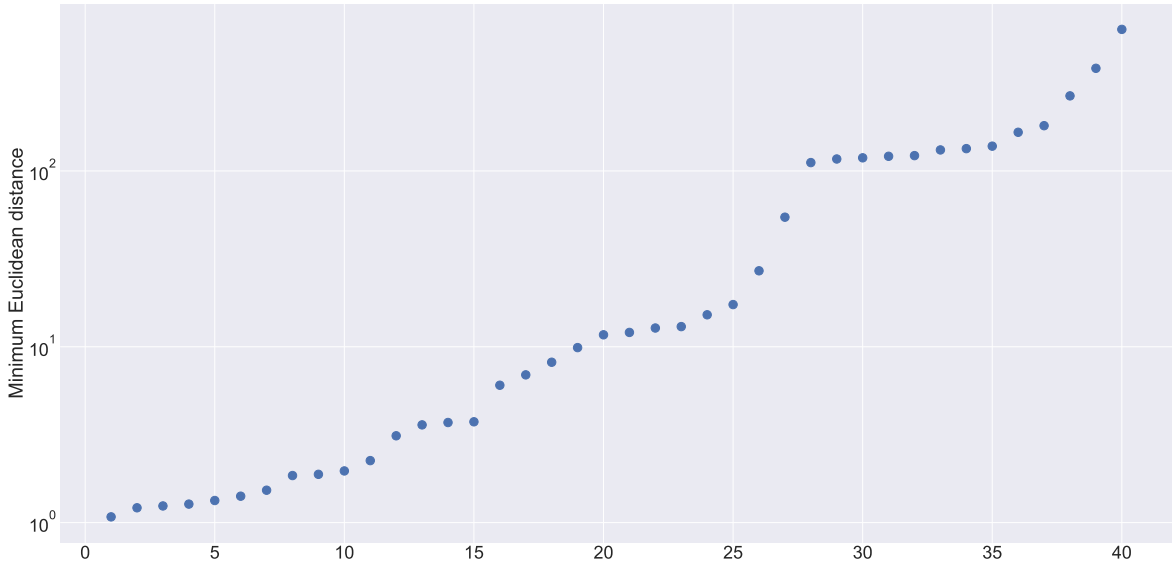


Figure 7 **The minimum Euclidean distance between a point in the approximate nondominated frontier generated by NSGA-II, and the one that dominates it in the approximate nondominated frontier generated by V1.**

a point in the approximate nondominated frontier generated by NSGA-II, and those that dominate it in the approximate nondominated frontier generated by V1. The horizontal line in this figure shows the instance index/number. Note that there are only 40 instances, and we have sorted them in non-decreasing order of their minimum Euclidean distance. We observe that the minimum Euclidean distance is between 1 to over 10^2, which clearly shows that the approximate nondominated frontier generated by NSGA-II must be far from the true nondominated frontier.

Figure 8 **The true nondominated frontier vs the approximate nondominated frontiers obtained by V1, V2, and V3 for an instance of bi-objective set packing problem.**

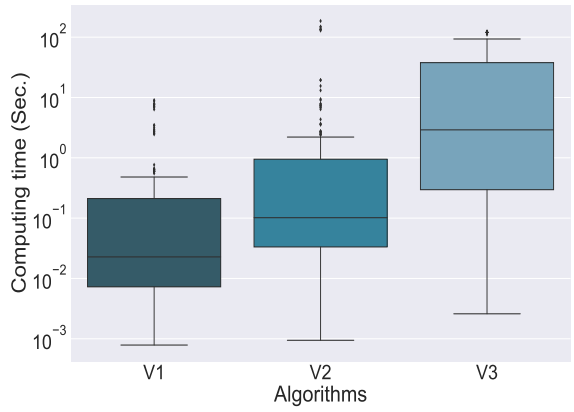
6.2. V1 vs V2 vs V3

Figure 8 shows the approximate nondominated frontiers obtained by V1, V2 and V3, and the true nondominated frontier for one of our instances of the bi-objective set packing problem. Observe that the approximate nondominated frontier of V2 is significantly better than V1, and V3 is significantly better than V2. The true nondominated frontier of this instance has 10 points, and 6 of them are obtained by V3.

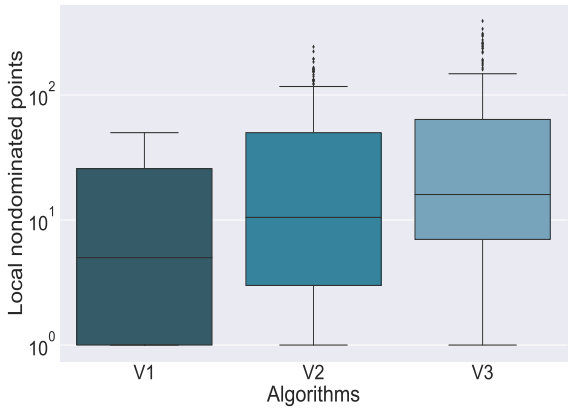
In this section, we impose a time limit of 120 seconds for all our experiments. Figure 9 shows the performance of V1, V2, and V3 for all 254 instances. In Figure 9a, we observe that, on average, both V1 and V2 are terminated in a fraction of a second, and V3 is terminated in less than 10 seconds. In Figure 9b, we observe that more points exist in the approximate nondominated frontier of the sophisticated versions, i.e., V2 and V3. Also, Figure 9c-9f show that the approximate nondominated frontier of V3 reaches a better value for the hypervolume gap, cardinality, coverage, and uniformity indicators in comparison to V2. Similarly, the approximate nondominated frontier of V2 reaches a better value for these indicators in comparison to V1. It is worth mentioning that Figure 9d shows that on average around 10% of true nondominated points are obtained by using V3.

6.3. V3 vs MDLS

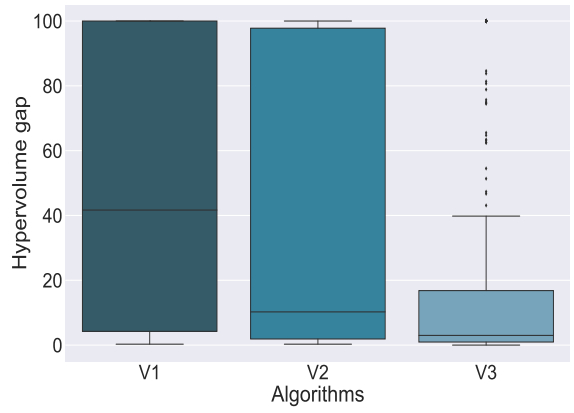
In this section, we compare the performance of V3 with MDLS (Tricoire 2012). For this comparison, we have used the C++ implementation of MDLS which is publicly available at `http://prolog.univie.ac.at/research/MDLS`. It is worth mentioning that this implementation is problem-dependent in a sense that some of its source/header files should be



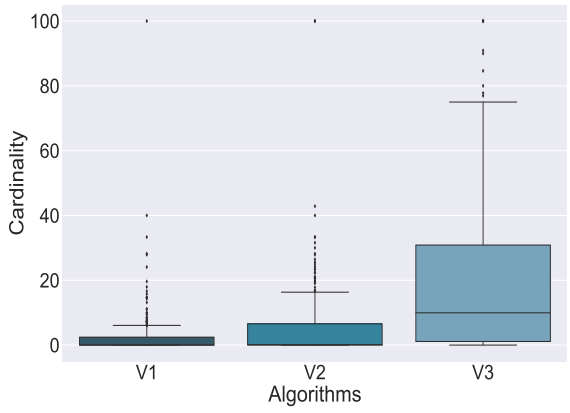
(a) Solution time.



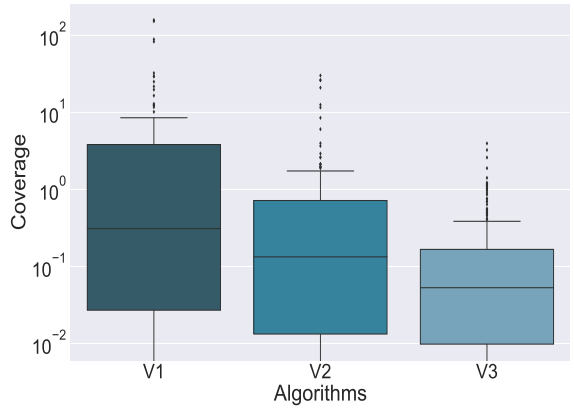
(b) The number of points.



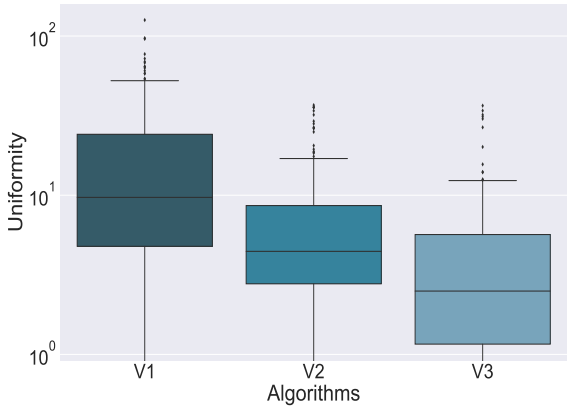
(c) Hypervolume gap.



(d) Cardinality.



(e) Coverage.



(f) Uniformity.

Figure 9 Performance of V1, V2, and V3.

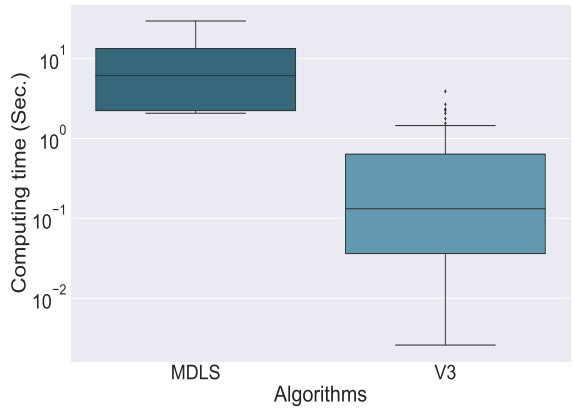
specifically defined for any problem that we are trying to solve. Fortunately, we are able to employ MDLS for 222 instances of our total 254 instances. This is because in the available implementation, the corresponding header/source files for solving instances of the one-dimensional knapsack problem, i.e., those with one constraint, and the two-dimensional knapsack problem, i.e., those with two constraints, and the set packing problem exist.

We use the default setting of MDLS in which it repeats for 10 runs for each instance, and reports the results of each run. For comparison purposes, we accumulate the results of all 10 runs for each instance. We again impose a time limit of 120 seconds for V3 but do not impose any time limit for the total run time of MDLS (with 10 runs) since it naturally terminates within 120 seconds for all 222 instances. Figure 10, 11 and 12 show the performance of V1, V2 and V3 on the instances of the one-dimensional knapsack problem, the two dimensional knapsack problem, and the set packing problem, respectively.

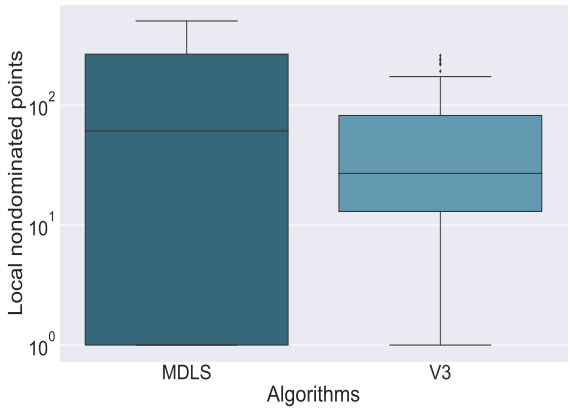
We observe that the average run time of V3 is smaller than MDLS for all three different classes of problems. However, in terms of the average uniformity, MDLS outperforms V3 for all three different classes of problems since it finds more local nondominated points. Overall, we see that,

- For the instances of the one-dimensional knapsack problem, both algorithms are competitive since V3 is better in terms of the hypervolume gap and cardinality, but MDLS is better in terms of the coverage and uniformity, on average.
- For the instances of the two-dimensional knapsack problem V3 dominates MDLS since it is better in terms of the hypervolume gap, cardinality and coverage, on average.
- For the instances of the set packing problem, MDLS outperforms V3 since it is better in terms of the hypervolume gap and cardinality, coverage, and uniformity, on average.

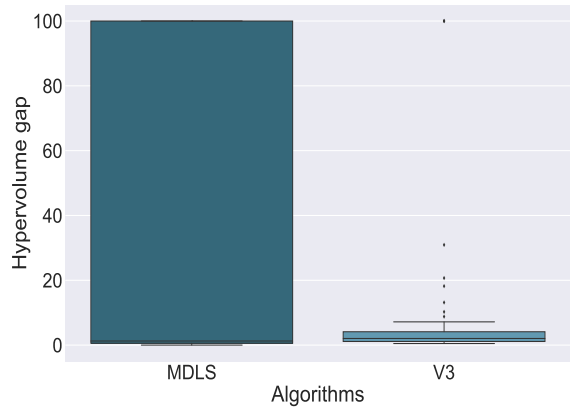
In summary, we observe that MDLS and V3 are competitive. We now make two final comments. (1) MDLS can be customized and applied to problems with more than two objectives. On the other hand, at the moment, our proposed heuristic can only be applied to problems with only two objective functions. However, this does not mean that the framework of our proposed algorithm cannot be extended to more than two objective functions. In other words, this basically requires further research. (2) MDLS is successfully applied to problems similar to the bi-objective traveling salesman problem, see for instance, bi-objective orienteering problem (Tricoire 2012). Obviously, there are different ways to formulate this kind of problems and some of them requires an exponential number of



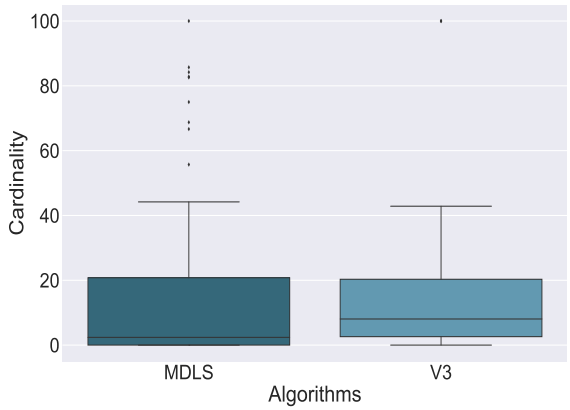
(a) Solution time.



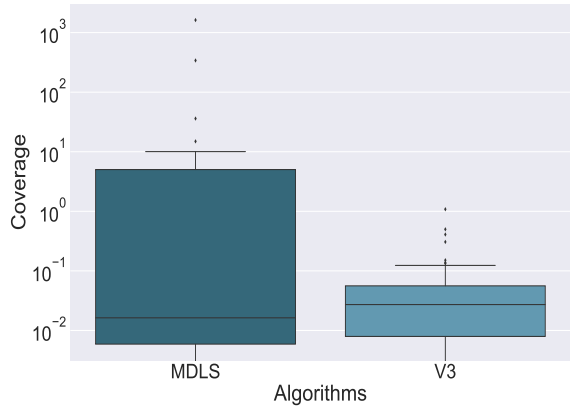
(b) The number of points.



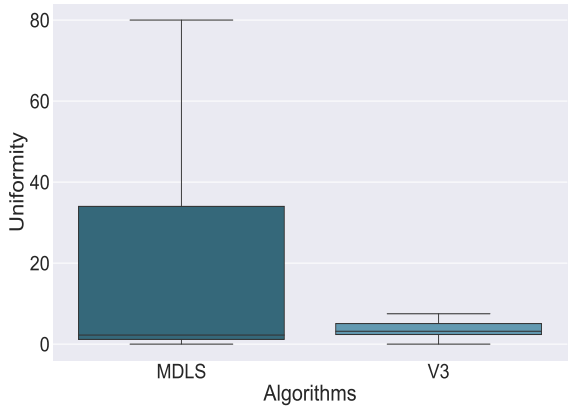
(c) Hypervolume gap.



(d) Cardinality.

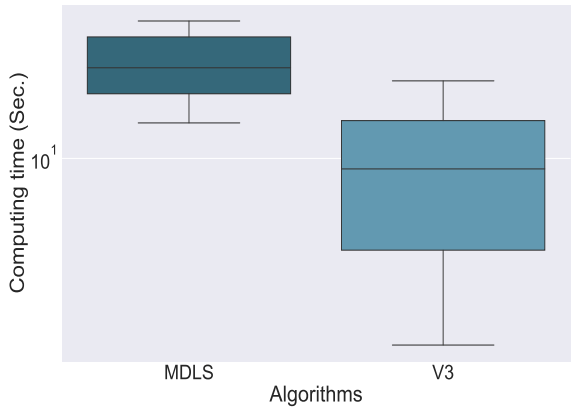


(e) Coverage.

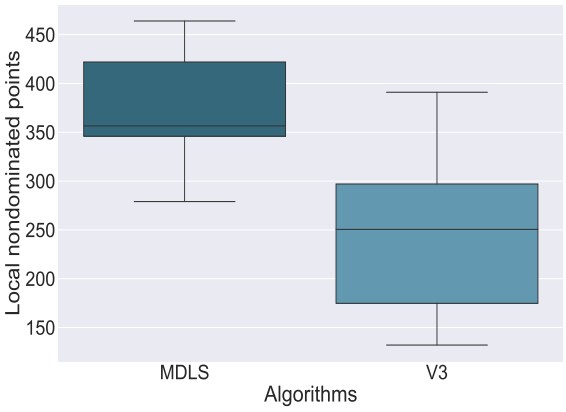


(f) Uniformity.

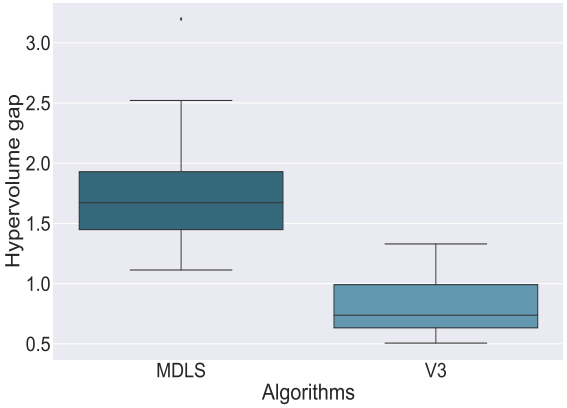
Figure 10 Performance of V3 and MDLS on the instances of the one-dimensional knapsack problem.



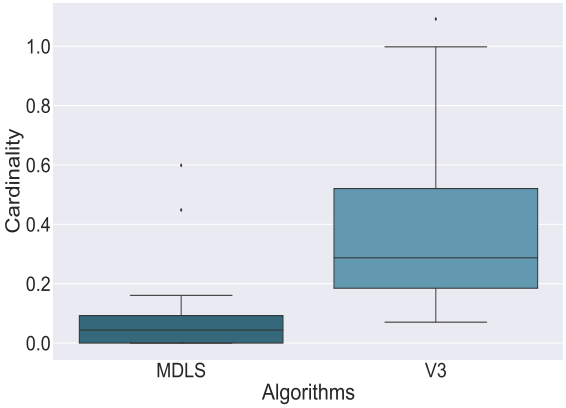
(a) Solution time.



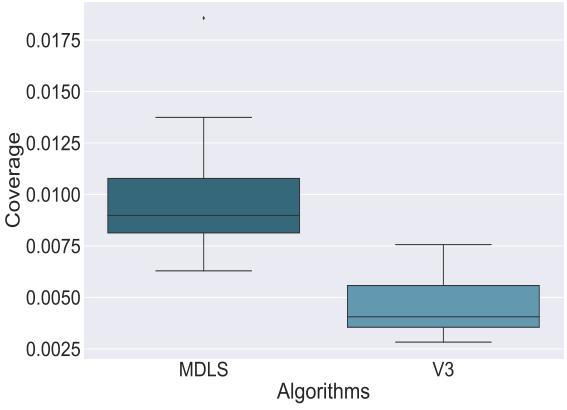
(b) The number of points.



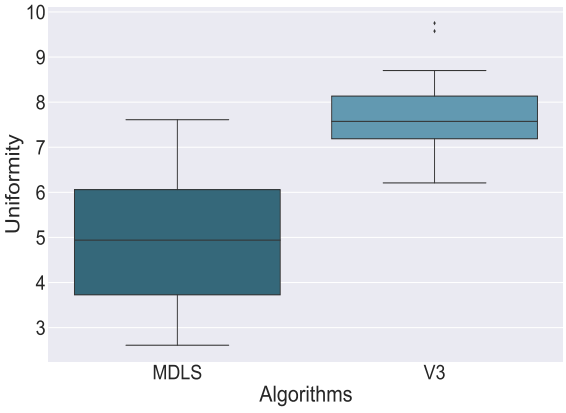
(c) Hypervolume gap.



(d) Cardinality.

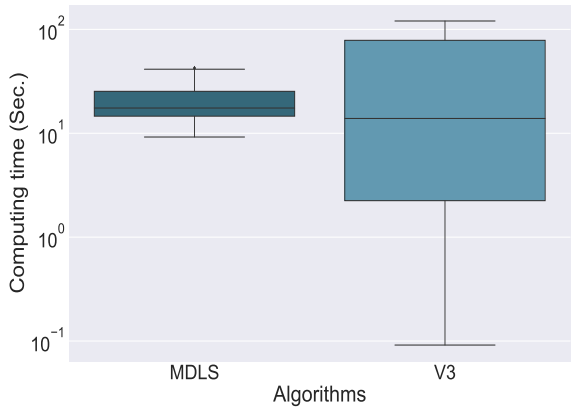


(e) Coverage.

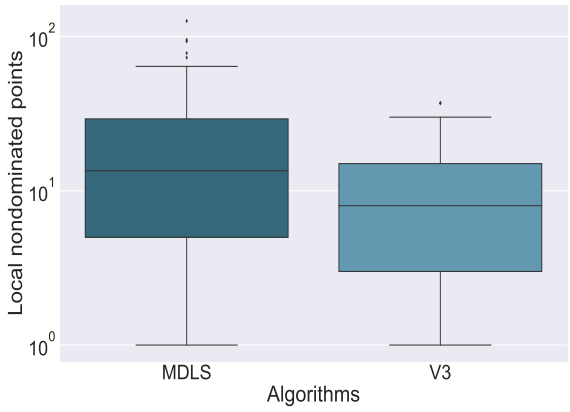


(f) Uniformity.

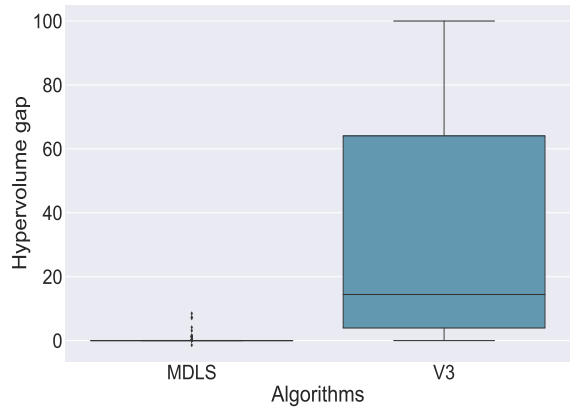
Figure 11 Performance of V3 and MDLS on the instances of the two-dimensional knapsack problem.



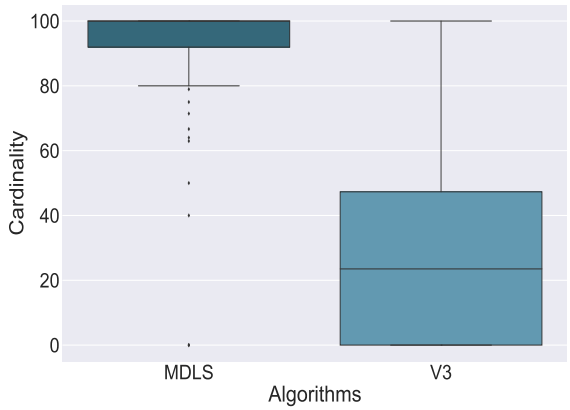
(a) Solution time.



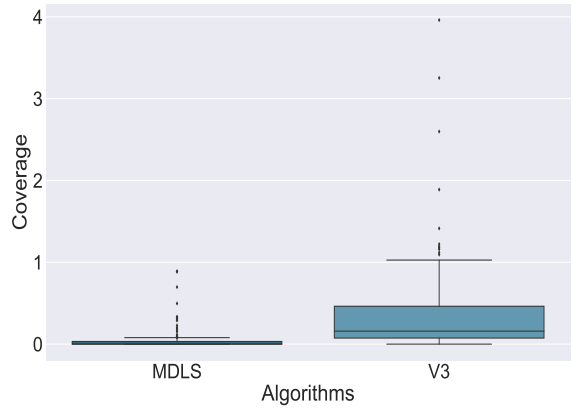
(b) The number of points.



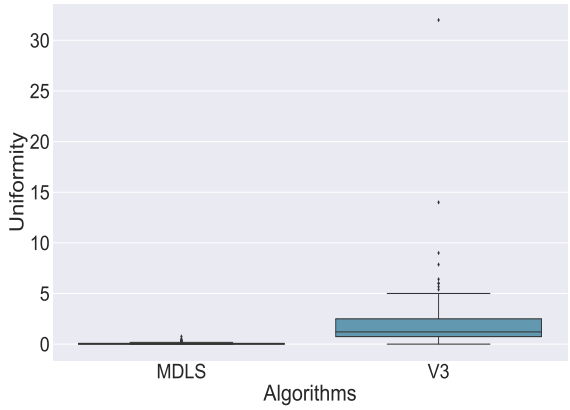
(c) Hypervolume gap.



(d) Cardinality.



(e) Coverage.



(f) Uniformity.

Figure 12 Performance of V3 and MDLS on the instances of the set packing problem.

constraints that cannot be directly handled by our algorithm. Consequently, the choice of the formulation can be quite important for the success of our proposed algorithm. Of courses, this observation is valid for any feasibility pump based heuristic.

6.4. Parallelization

Nowadays, most computers have multiple cores and computations can be divided over multiple threads, which can substantially reduce run times. Consequently, since the proposed approach can naturally exploit parallelism, we now show the performance of V3 when one or two or three or four threads are available. In this section, we impose a time limit of 120 seconds for all our experiments.

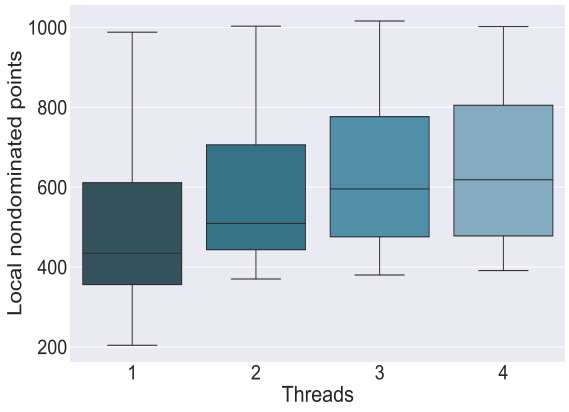
To conduct the experiments in this section, we randomly generate 12 classes of large instances by choosing $m \in \{25, 50, 75, 100\}$ and $n \in \{1,000, 1,500, 2,000\}$. Note that m is the number of constraints and n is the number of (binary) variables. Since we generate five instances for each class, a total of 60 instances is used in this section. We draw c_j^1 and c_j^2 randomly from the discrete uniform distribution from the interval $[-100, -1]$ for all $j \in \{1, \ldots, n\}$. We also select the entries of matrix A randomly from the discrete uniform distribution from the interval $[0, 100]$. Finally, for all $i \in \{1, \ldots, m\}$, we set $b_i = \lceil \frac{\sum_{j=1}^{n} a_{ij}}{2} \rceil$ where a_{ij} is the entry corresponding to row i and column j of matrix A.

For any given instance, let $\mathcal{Y}_N^U$ be the union of the approximate nondominated frontier generated by V3 when using 1, 2, 3, and 4 threads. Since for the instances used in this section, we do not have the true nondominated frontier, we treat $\textsc{Pareto}(\mathcal{Y}_N^U)$ as the true nondominated frontier, i.e., $\mathcal{Y}_N$. This enables us to use the proposed quality indicators for our analysis.

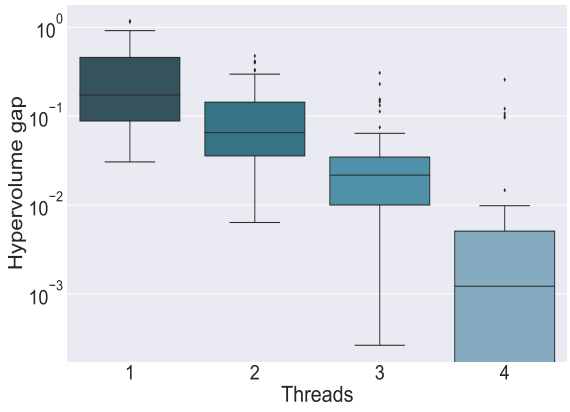
Figure 13 shows the performance of V3 on multiple threads. It is evident that the approximate nondominated frontier generated by V3 improves with respect to all proposed quality indicators as we increase the number of threads. More precisely, Figures 13a shows that more points are found in the approximate frontier as we increase the number of threads. Also, Figure 13b-13e show that the approximate nondominated frontier has reached a better value for the hypervolume gap, cardinality, coverage, and uniformity indicators as we increase the number of threads.

7. Final remarks

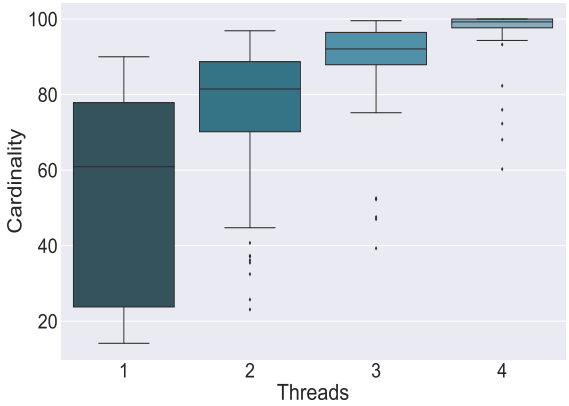
We presented a new heuristic algorithm for computing an approximate nondominated frontier of any bi-objective pure integer linear program. The proposed algorithm uses



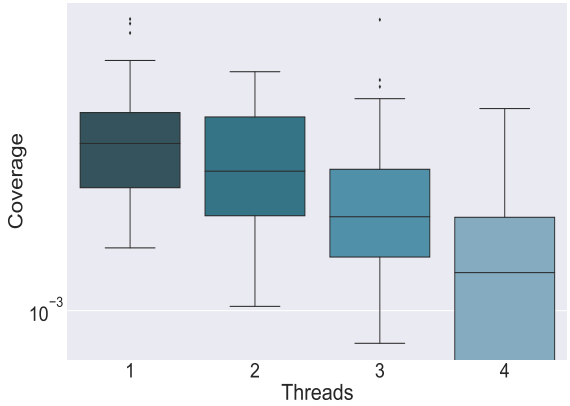
(a) The number of points.



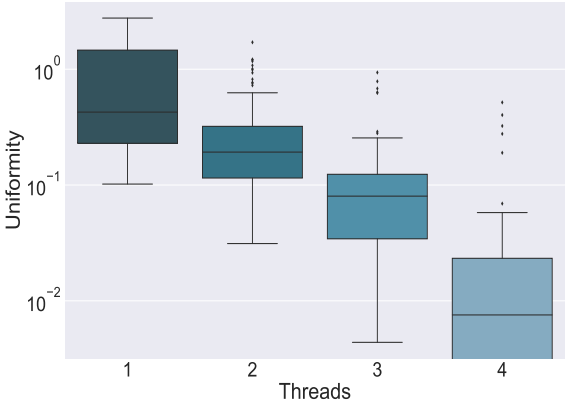
(b) Hypervolume gap.



(c) Cardinality.



(d) Coverage.



(e) Uniformity.

Figure 13 Performance of V3 using multiple threads.

the underlying ideas of several exact/heuristic algorithms in the literature of both single and bi-objective integer linear programs including the perpendicular search method, the feasibility pump heuristic, the local search approach, and the weighted sum method. The feasibility pump heuristic has been successfully used in the literature of single-objective optimization. Also, the fact that the commercial exact single-objective optimization solvers (such as CPLEX) are using it, shows that this heuristic can enhance the exact solution approaches. To the best of our knowledge, we are the first authors introducing a customized variant of this heuristic for multi-objective optimization. We hope the simplicity and the efficacy of our method encourage more researchers to work on the feasibility pump based heuristics for multi-objective optimization, and possibly combining them with the exact solutions approaches (to boost them) just like what happened in the world of single-objective optimization.

References

Achterberg T, Berthold T (2007) Improving the feasibility pump. *Discrete Optimization* 4(1):77 – 86, mixed Integer ProgrammingIMA Special Workshop on Mixed-Integer Programming.

Aneja YP, Nair KPK (1979) Bicriteria transportation problem. *Management Science* 27:73–78.

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

Boland N, Charkhgard H, Savelsbergh M (2015b) A criterion space search algorithm for biobjective mixed integer programming: The triangle splitting method. *INFORMS Journal on Computing* 27(4):597–618.

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

Boland N, Charkhgard H, Savelsbergh M (2016b) The quadrant shrinking method: A simple and efficient algorithm for solving tri-objective integer programs. *European Journal of Operational Research* Available online.

Boland NL, Eberhard AC, Engineer FG, Fischetti M, Savelsbergh MWP, Tsoukalas A (2014) Boosting the feasibility pump. *Mathematical Programming Computation* 6(3):255–279.

Captivo ME, Clımaco J, Figueira J, Martins E, Santos JL (2003) Solving bicriteria 01 knapsack problems using a labeling algorithm. *Computers & Operations Research* 30(12):1865 – 1886.

Castillo Tapia M, Coello Coello CA (2007) Applications of multi-objective evolutionary algorithms in economics and finance: A survey. *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, 532–539.

Chalmet LG, Lemonidis L, Elzinga DJ (1986) An algorithm for bi-criterion integer programming problem. *European Journal of Operational Research* 25:292–300.

Coello Coello CA, Lamont GB, Van Veldhuizen DA (2007) *Evolutionary Algorithms for Solving Multi-Objective Problems* (New York: Springer).

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

Dächert K, Klamroth K (2014) A linear bound on the number of scalarizations needed to solve discrete tricriteria optimization problems. *Journal of Global Optimization* 1–34.

Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation* 6(2):182–197.

Degoutin F, Gandibleux X (2002) Un retour dexpériences sur la résolution de problemes combinatoires bi-objectifs.

Delorme X, Gandibleux X, Degoutin F (2010) Evolutionary, constructive and hybrid procedures for the bi-objective set packing problem. *European Journal of Operational Research* 204(2):206 – 217.

Ehrgott M, Gandibleux X (2004) Approximative solution methods for multiobjective combinatorial optimization. *Top* 12(1):1–63.

Fischetti M, Glover F, Lodi A (2005) The feasibility pump. *Mathematical Programming* 104(1):91–104.

Gandibleux X, Freville A (2000) Tabu search based procedure for solving the 0-1 multiobjective knapsack problem: The two objectives case. *Journal of Heuristics* 6(3):361–383.

Isermann H (1977) The enumeration of the set of all efficient solutions for a linear multiple objective program. *Operational Research Quarterly* 28(3):711–725.

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

Köksalan M, Lokman B (2014) Finding nadir points in multi-objective integer programs. *Journal of Global Optimization* 1–23.

Lei D (2009) Multi-objective production scheduling: a survey. *International Journal of Advanced Manufacturing Technology* 43(9):926–938.

Lin S, Kernighan BW (1973) An effective heuristic algorithm for the traveling-salesman problem. *Operations Research* 21(2):498–516.

Lubin M, Dunning I (2015) Computing in operations research using Julia. *INFORMS Journal on Computing* 27(2):238–248.

Lust T, Teghem J (2010) The multiobjective traveling salesman problem: A survey and a new approach. Coello Coello CA, Dhaenens C, Jourdan L, eds., *Advances in Multi-Objective Nature Inspired Computing*, volume 272 of *Studies in Computational Intelligence*, 119–141 (Springer Berlin Heidelberg).

Marler R, Arora J (2004) Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization* 26(6):369–395.

Özlen M, Burton BA, MacRae CAG (2013) Multi-objective integer programming: An improved recursive algorithm. *Journal of Optimization Theory and Applications* 10.1007/s10957-013-0364-y.

Papadimitriou CH, Yannakakis M (2000) On the approximability of trade-offs and optimal access of web sources. *41st Annual Symposium on Foundations of Computer Science. Proceedings*, 86–92.

Paquete L, Schiavinotto T, Stützle T (2007) On local optima in multiobjective combinatorial optimization problems. *Annals of Operations Research* 156(1):83.

Sayın S (2000) Measuring the quality of discrete representations of efficient sets in multiple objective mathematical programming. *Mathematical Programming* 87(3):543–560.

Soylu B (2015) Heuristic approaches for biobjective mixed 0-1 integer linear programming problems. *European Journal of Operational Research* 245(3):690 – 703.

Tricoire F (2012) Multi-directional local search. *Computers & Operations Research* 39(12):3089 – 3101.

Zhou A, Qu BY, Li H, Zhao SZ, Suganthan PN, Zhang Q (2011) Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation* 1(1):32–49.

Zitzler E, Brockhoff D, Thiele L (2007) The hypervolume indicator revisited: On the design of Pareto-compliant indicators via weighted integration. Obayashi S, Deb K, Poloni C, Hiroyasu T, Murata T, eds., *Evolutionary Multi-Criterion Optimization*, volume 4403 of *Lecture Notes in Computer Science*, 862–876.

Zitzler E, Thiele L, Laumanns M, Fonseca C, Grunert da Fonseca V (2003) Performance assessment of multiobjective optimizers: an analysis and review. *Evolutionary Computation, IEEE Transactions on* 7(2):117–132.