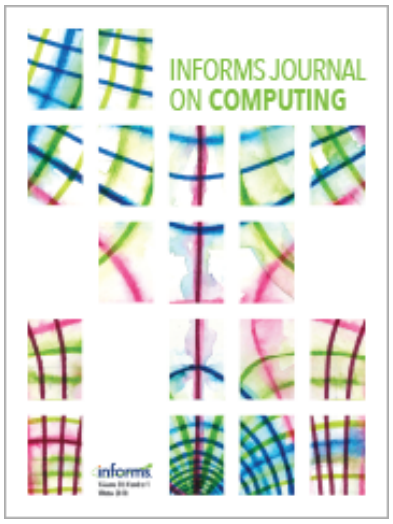
Nondominated Set for Multiobjective Discrete Optimization

Multiobjective optimization is one of the historical field of expertise of LAMSADE. This paper addresses multiobjective problems where the set of solutions is defined combinatorially. Indeed, many combinatorial optimization problems require taking into account multiple criteria. In this context, an important issue is to determine the set of efficient solutions (also called Pareto-optimal solutions) or more precisely the set of nondominated points, which corresponds to the images of these solutions in the objective space. A main challenge, both from the decision making and computational viewpoints, is the potentially huge size of the nondominated set which can grow exponentially with the size of the instances for multiobjective combinatorial optimization problems. Many generic algorithms were proposed in the literature to address this challenge. Most of these algorithms probe the search space by making iterative calls to a MIP solver in order to find new nondominated points, or prove that some parts of the search space do not contain any new nondominated point. The computational efficiency of such algorithms critically depends on the following factors: (1) limiting the number of calls to the solver while guaranteeing exactness and completeness of the generated nondominated set (2) easing the task of the solver, by providing a feasible solution (warm start) whenever possible and avoiding calls to infeasible instances, which usually require a larger exploration for the solver.

Our algorithm is based on our previous formalization and study of the concept of search region which characterizes the part of the objective space where new non-dominated points may lie [1]. For this purpose, our algorithm splits the search region into zones which can be investigated separately by solving an integer program. We also propose refinements, which provide extra information on several zones, allowing us to detect, and discard, empty parts of the search region without checking them by solving the associated integer programs. This results in a limited number of calls to the solver. Moreover, we can provide a feasible starting solution before solving every program, which significantly reduces the time spent for each resolution. Finally we are able to guarantee that none of the calls is infeasible. The resulting algorithm is fast and simple to implement. It is compared with previous state-of-the-art algorithms on various types of instances and is shown to outperform them significantly on the experimented problem instances. An implementation of the algorithm is available at the following address: https://www.lamsade.dauphine.fr/ vdp/tv19/

References

[1] K. Dächert, K. Klamroth, R. Lacour, and Daniel Vanderpooten. Efficient computation of the search region in multi-objective optimization. *European Journal of Operational Research*, 260(3), 2017.



INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Enumeration of the Nondominated Set of Multiobjective Discrete Optimization Problems

Satya Tamby, Daniel Vanderpooten

To cite this article:
Satya Tamby, Daniel Vanderpooten (2020) Enumeration of the Nondominated Set of Multiobjective Discrete Optimization Problems. INFORMS Journal on Computing

Published online in Articles in Advance 04 Jun 2020

. https://doi.org/10.1287/ijoc.2020.0953

http://pubsonline.informs.org/journal/ijoc

INFORMS JOURNAL ON COMPUTING
Articles in Advance, pp. 1–14
ISSN 1091-9856 (print), ISSN 1526-5528 (online)

Enumeration of the Nondominated Set of Multiobjective Discrete Optimization Problems

Satya Tamby,[a] **Daniel Vanderpooten**[a]

[a] Université Paris Dauphine, PSL Research University, CNRS, UMR 7243, LAMSADE, 75016 Paris, France

Contact: satya.tamby@dauphine.fr, https://orcid.org/0000-0002-3310-272X (ST); daniel.vanderpooten@lamsade.dauphine.fr, https://orcid.org/0000-0002-5934-0603 (DV)

Received: November 5, 2018
Revised: May 20, 2019; November 9, 2019; December 2, 2019
Accepted: December 4, 2019
Published Online in Articles in Advance: June 4, 2020

https://doi.org/10.1287/ijoc.2020.0953

Abstract. In this paper, we propose a generic algorithm to compute exactly the set of nondominated points for multiobjective discrete optimization problems. Our algorithm extends the ε-constraint method, originally designed for the biobjective case only, to solve problems with two or more objectives. For this purpose, our algorithm splits the search space into zones that can be investigated separately by solving an integer program. We also propose refinements, which provide extra information on several zones, allowing us to detect, and discard, empty parts of the search space without checking them by solving the associated integer programs. This results in a limited number of calls to the integer solver. Moreover, we can provide a feasible starting solution before solving every program, which significantly reduces the time spent for each resolution. The resulting algorithm is fast and simple to implement. It is compared with previous state-of-the-art algorithms and is seen to outperform them significantly on the experimented problem instances.

History: Accepted by David Woodruff, (former) Area Editor for Software Tools.

Keywords: multiobjective optimization • combinatorial optimization • nondominated points • ε-constraint

1. Introduction

Real-world problems often involve several conflicting criteria. In this case, solutions of interest are *efficient* solutions, which have the property that any improvement on one criterion leads to a decay on at least another one (*Pareto optimality*). The images of such solutions in the objective space are referred to as *nondominated* points. A standard goal in multiobjective optimization consists of generating the set of nondominated points and providing for each point a corresponding efficient solution. Because of the size of the final nondominated set, which can be huge, multiobjective optimization problems are significantly harder to solve than their single-objective version.

In this paper, we focus on multiobjective combinatorial optimization problems and more generally on multiobjective discrete problems. Several algorithms have been designed to solve multiobjective variants of specific problems—for example, the shortest path problem (see Martins 1984) or the knapsack problem (see Bazgan et al. 2009). However, these algorithms rely on specific properties of the problem, and it is not possible to extend them to other problems.

Other approaches are multiobjective extensions of implicit enumeration methods (branch and bound and dynamic programming). These methods operate in the decision space, completing progressively partial solutions by setting iteratively the decision variables (through branching) and trying to restrict the enumeration by pruning partial solutions whose extensions can be proved not to give rise to new nondominated points (usually through bounding). Although these enumeration schemes are general, the definition of branching strategies and the determination of tight bounds depend on the specific problem to be solved. We refer the interested reader to the recent survey about multiobjective branch and bound (Przybylski and Gandibleux 2017), and the previously mentioned papers about the multiobjective shortest path and knapsack can be seen as applications of multiobjective dynamic programming.

In this paper we consider generic algorithms that do not depend in their description on the specific problem to be solved (they only need to call as a black box a solver able to solve a single-objective version of the specific problem usually augmented with some constraints). Such algorithms operate in the criterion space.

Generic algorithms usually resort to integer programming solvers to be suitable for multiobjective discrete optimization problems because most of them involve linear constraints and objectives. They iteratively solve single-objective programs over the *search region* (i.e., the subset of the objective space that potentially contains the remaining nondominated points). When a new point is found, the search region is updated by eliminating the part of this region dominated by this point, and the procedure is repeated until the search region becomes empty.

In the biobjective case, the ε-constraint method (Chankong and Haimes 1983) works well in practice. This method iteratively generates nondominated points by optimizing the first objective, whereas the second objective is constrained to be better than the nondominated point found at the previous iteration. Thus, each step yields a new nondominated point except for the last one, which leads to an infeasible program, showing that all points have been found. Generalizing the ε-constraint method to more than two objectives is a challenging issue for which several algorithms have been proposed. In the following, we first give the intuition behind the different algorithmic strategies, and then we describe more precisely some representative algorithms.

In the first type of approach (Klein and Hannan 1982, Sylva and Crema 2004), the original integer program is iteratively augmented by disjunctive constraints that impose that the next solution must improve at least one objective with respect to all previously obtained points. Thus, as in the ε-constraint method, each optimization yields a new nondominated point except for the last one. However, because the number of constraints and variables grows each time a new point is found, the program gets harder to solve over the iterations, and the algorithm becomes quickly impractical when solving even rather small instances.

In order to circumvent these difficulties, other approaches split the search region into a union of zones, each one being delimited by a local upper bound on the objectives (assuming they are to be minimized). Each zone can then be explored using a budget-constrained program where one constraint per objective is added to the original problem. Empty zones are discarded, and if a new point is found, then the zones containing this point are subdivided in order to avoid finding the same point again. Explorations are performed until all the remaining zones are empty. The main advantage of such approaches is that the size of the budget-constrained programs remains constant, which does not alter the resolution over the iterations. However, unlike in the first type of approach, the number of programs to be solved is substantially larger than the number of nondominated points. Indeed, many zones to be explored are actually empty, which is checked when the corresponding program is infeasible.

The main challenge for these methods is to minimize the total number of programs to be solved. Moreover, in order to get a practically efficient method, two important issues must be taken into account. First, infeasible programs should be avoided because integer programming solvers usually require more time to prove infeasibility than to find an optimal solution (because of the ability to cut the search tree as soon as a feasible solution is found). Therefore, even if the zone to be explored is empty, it is practically more efficient to find a way to prove this by solving some feasible program. Second, for feasible programs, providing a feasible solution to the solver also speeds up the resolution. The practical impact of these two issues is also pointed out in Boland et al. (2016).

Algorithms differ in the way local upper bounds are computed and used. We can distinguish two main strategies depending on whether the bounds constrain all of the p objectives (full-dimensional bounds) or only a subset (projected bounds), usually on $p-1$ objectives. Strategies relying on full-dimensional bounds investigate, at each iteration, one search zone precisely delimited on each objective. When a zone is empty, this results in an infeasible program. This case may occur quite frequently. In such approaches, once a nondominated point is generated, it will not be returned at a later iteration because of the exact definition of the search region with full-dimensional bounds. By contrast, strategies using projected bounds may return a point that does not belong to the explored zone, allowing us to explore several zones simultaneously. However, as a negative consequence, the same point may be generated several times at different iterations. Indeed, because the search zone that is explored is not bounded on the omitted objective, the enumerated point may not belong to the search region. As a consequence, an additional test is required to detect potential duplicates in the final nondominated set.

Algorithms also differ on the way the zones are explored or, more precisely, in the type of objective function used when solving the budget-constrained programs. A main distinction is between algorithms optimizing a linear combination of the objectives, with strictly positive weights, and algorithms favoring one objective (i.e., performing a lexicographic optimization). Optimizing a linear combination can directly be performed using a single program that returns an arbitrary nondominated point in the zone when it exists. Lexicographic optimization needs an additional effort: it commonly involves two successive programs optimizing the favored objective first and, in a second stage, a combination of the other objectives to ensure that the final point is nondominated. However, the returned point is not arbitrary anymore because it guarantees there is no point being better on the favored objective over the explored zone. This extra information can be used to prove the emptiness of other zones that do not need to be explored. This can significantly reduce the number of iterations needed to compute the set of nondominated points. We now briefly present some algorithms relying on budget-constrained programs.

Sylva and Crema (2008) propose an improvement of their previous algorithm described in Sylva and Crema (2004). Instead of using a single integer

program that performs the exploration over the whole search region, they use two programs: a master budget-constrained program and an auxiliary integer program that aims at defining the full-dimensional bounds to be used by the master program. However, the auxiliary program grows at each iteration, when new points are discovered or when zones are shown to be empty, and becomes huge and hard to solve. Instead of using an auxiliary program, Lokman and Köksalan (2013) propose to decompose the large program used in Sylva and Crema (2004) into several constant-size programs. However, the number of programs needed to be solved to obtain a new point increases along the iterations. For these reasons, solving large instances seems to be difficult with these approaches.

Mavrotas (2009) propose to characterize the search region as a $p-1$-dimensional hypergrid. The approach computes first the ranges of the values taken by $p-1$ objectives and then splits the search space into equal cells of unit size in each dimension. Then, each cell is explored independently, which amounts to considering all possible combinations of bounds on each objective. Several improvements of this approach have been proposed (see Mavrotas and Florios 2013 and Zhang and Reimann 2014) to detect cells that are empty or contain a redundant point without solving an integer program. Nevertheless, the approach remains quite enumerative.

Instead of defining the cells prior to the enumeration, the approach proposed by Özlen and Azizoğlu (2009) dynamically adapts it depending on the components of the points founds. However, even if a significant improvement has been proposed in Özlen et al. (2014) in order to avoid some redundant problems to be solved, a lot of points are generated several times. Moreover, in order to ensure that the generated points are nondominated, the authors optimize a weighted sum of the objectives involving very small weights on all but one objective, which can induce numerical instability.

Dhaenens et al. (2010) propose an iterative algorithm relying on the observation that some of the nondominated points in the global problem must also be nondominated in subproblems involving only a subset of the objectives. After these points have been computed, the remaining search region is split into zones that are explored separately. Specific attention has been paid for parallelism, but the number of multiobjective integer problems required to be solved grows exponentially with the number of objectives. Moreover, some points can be enumerated several times if they are nondominated in several subproblems.

Kirlik and Sayin (2014) propose an algorithm that stores the search region as an explicit list of boxes (instead of zones) defined using both upper and lower projected bounds on $p-1$ objectives (based on the characterization initially proposed by Laumanns et al. 2005). With this explicit characterization as a list, the search region can be incrementally updated when a new point is found: this is achieved by browsing the list and splitting each box the new point belongs to, without having to recompute the entire list. However, even if this method solves triobjective problems relatively fast, it struggles when the number of objectives grows. Indeed, because each box is both upper and lower bounded, splitting it when a point is found in this box induces a huge number of new boxes, and storing them leads to memory space problems when solving large instances.

Klamroth et al. (2015) precisely define the concept of search region as an explicit covering of the search space, which relies on full-dimensional zones, referred to as search zones. It is then still possible to perform an incremental update on this list. Moreover, the characterization of the search region is exact: each nondominated point is enumerated exactly once, and it is not necessary to filter the duplicates in the final set. This results in a simple generic algorithmic scheme, presented in Klamroth et al. (2015), consisting of exploring iteratively the search zones and updating the search region depending on whether or not the explored search zone contains a point. Although the number of search zones is significantly smaller than the number of boxes in Kirlik and Sayin (2014), the number of infeasible problems to be solved can be substantial.

Boland et al. (2017) propose to explore several zones simultaneously using a single program. The exploration of such shapes is performed by introducing p disjunctive constraints to the program. However, even if the global number of iterations is reduced, such programs involving disjunctive constraints are harder to solve than standard budget-constrained programs without disjunctive constraints.

Our goal in this paper consists of reducing not only the total number of budget-constrained programs needed to be solved but also the number of infeasible programs. To do so, we propose an algorithm relying on full-dimensional search zones to maintain an exact characterization of the search region as in the algorithmic scheme of Klamroth et al. (2015). However, the integer programs to be solved are budget-constrained programs using projections of these bounds, allowing us to guarantee that all the programs are feasible and to derive information on several search zones simultaneously. For this purpose, we propose conditions under which some search zones can be proved to be empty without checking them by solving an integer program. Moreover, by dynamically changing the omitted objective depending on the explored zone, we can provide, in constant time, a feasible solution for each exploration. To the best of our knowledge, this is the first algorithm using simultaneously full-

dimensional and projected search zones, exploiting the advantages of each strategy. The resulting algorithm, experimented on instances of multiobjective knapsack and assignment problems, clearly outperforms other state-of-the-art algorithms.

The rest of this paper is structured as follows. Section 2 introduces basic concepts and notations and focuses on the central concept of search region. Then, in Section 3, the core of our algorithm is presented, followed by several technical improvements. Computational experiments are reported in Section 4. Conclusions are provided in a final section. The progression of our algorithm on an illustrative example is described in the appendix.

2. Background

2.1. Basic Concepts and Notations

We consider multiobjective optimization problems with p objective functions to be minimized over a discrete, nonempty, set X of feasible solutions:

$$\begin{cases} \min & f_1(x), \ldots, f_p(x) \\ \text{s.t.} & x \in X. \end{cases} \tag{1}$$

An element of $x \in X$ is called a *feasible solution*, and the corresponding performance vector, $y = (f_1(x), \ldots, f_p(x))$, is called a *feasible point*. Therefore, the objective space is $\mathbb{R}^p$, and $Y = f(X) \subset \mathbb{R}^p$ is the set of the images of all feasible solutions in the objective space. We assume that Y is bounded. Problem (1) can be restated in the objective space as

$$\begin{cases} \min & (y_1, \ldots, y_p) \\ \text{s.t.} & y \in Y. \end{cases} \tag{2}$$

In the following, we will state all our problems in the objective space, for the sake of conciseness.

Given a point $y \in \mathbb{R}^p$, we denote by y_{-k} its orthogonal projection on the subspace $\mathbb{R}^{p-1}$ (i.e., the $p-1$-dimensional vector where the k^{th} component has been omitted):

$$y_{-k} = (y_1, \ldots, y_{k-1}, y_{k+1}, \ldots, y_p).$$

Several partial orders can be defined on $\mathbb{R}^p$. Let y and y' in $\mathbb{R}^p$. We have

$$\begin{aligned} y \leqq y' &\Leftrightarrow y_i \leq y'_i, \ \forall i \in \{1, \ldots, p\}, \\ y \leq y' &\Leftrightarrow y \leqq y' \text{ and } y \neq y', \\ y \prec y' &\Leftrightarrow y_i < y'_i, \ \forall i \in \{1, \ldots, p\}. \end{aligned}$$

We refer to these relations as *weak dominance*, (*Pareto*) *dominance*, and *strict dominance*, respectively. A solution is called *efficient* (respectively, *weakly efficient*) if its image is not dominated (respectively, not strictly dominated). Let Y_{ND} be the set of points that are nondominated. In this work, we propose an algorithm to generate Y_{ND} and provide one of the corresponding efficient solutions for each point of this set.

We denote by $z(y)$ (respectively, $d(y)$) the set of points strictly dominating y(respectively, dominated by y):

$$\begin{aligned} z(y) &= \{y' \in \mathbb{R}^p, y' \prec y\}, \\ d(y) &= \{y' \in \mathbb{R}^p, y \leqq y'\}. \end{aligned}$$

Let y^I be the ideal point of Y, defined by $y^I_i = \min_{y \in Y}\{y_i\}$, $i \in \{1, \ldots, p\}$. Note that y^I provides a lower bound on each criterion and can be obtained by solving p programs minimizing each of the p criteria. Moreover, let M be an upper bound on each criterion; M can be the largest value it is possible to represent on a computer or determined using p maximization problems over the feasible set.

Finding a nondominated point in a zone is performed by optimizing a strongly monotone function.

Definition 1. A function $\phi : Y \to \mathbb{R}$ is *strongly monotone* if and only if for all $(y, y') \in Y^2$, $y \leq y'$ implies $\phi(y) < \phi(y')$.

Standard strongly monotone functions are as follows:

- *The weighted sum*: $\phi(y) = \sum_{i=1}^{p} \lambda_i y_i$ with, $\lambda_i > 0$, $\forall i \in \{1, \ldots, p\}$.
- *Lexicographic*: $\phi(y) = \text{lexmin}\{y_k, \sum_{\substack{i=1 \\ i \neq k}}^{p} \lambda_i y_i\}$, $\lambda_i > 0$, $\forall i \in \{1, \ldots, p\}$, $i \neq k$.

The following integer program generates a nondominated point strictly dominating the bound u when a feasible point exists in zone $z(u)$.

Proposition 1. *Let $u \in \mathbb{R}^p$ and $\phi : Y \to \mathbb{R}$ be a strongly monotone function. Consider the following program:*

$$P(u) \begin{cases} \min & \phi(y) = \phi(y_1, \ldots, y_p) \\ \text{s.t.} & y \in Y, \\ & y_i < u_i \qquad\qquad i \in \{1, \ldots, p\}. \end{cases}$$

Let y^* be an optimal solution of program $P(u)$, if it exists. Then, we have $y^* \in Y_{ND} \cap z(u)$.

Proof. Assuming by contradiction that y^* is dominated, there exists $y' \in Y$ such that $y' \leq y^*$. Because we have $y^* \prec u$, we get $y' \prec u$. Thus, y' is feasible for program $P(u)$. Moreover, we have $\phi(y') < \phi(y^*)$ because ϕ is strongly monotone, contradicting the optimality of y^*. □

Note that the p constraints $y_i < u_i$ for $i \in \{1, \ldots, p\}$ cannot be written in a linear program because they involve strict inequalities. We use $y_i \leq u_i - \varepsilon_i$ instead, with ε_i smaller than the smallest difference between the performance of two different points on criterion i. When instances involve integer values only (which is a common assumption in multiobjective discrete optimization), we can choose, for example, $\varepsilon_i = 0.5$, $i \in \{1, \ldots, p\}$.

Finally, given a program P to be solved, we denote by $F(P)$ its associated feasible domain.

2.2. Search Region

2.2.1. Definition. Following Klamroth et al. (2015) and Dächert et al. (2017), we first provide a formal definition of the *search region* that corresponds to the part of the objective space that may contain nondominated points that have not been generated so far.

Definition 2. Given a set N of points, the *search region associated with* N, referred to as $S(N)$, is defined by

$$S(N) = \mathbb{R}^p \setminus \bigcup_{y \in N} d(y) = \{y \in \mathbb{R}^p : \forall y' \in N, y' \not\leqq y\}.$$

Although N usually consists of nondominated points previously generated, it might include any point not even feasible. Observe also that for any point $y \in N$, we have $y \notin S(N)$. Thus, by definition, the search region excludes points already generated.

Generic algorithms iteratively *explore* the search region, or a superset of this region, using the integer program described in Proposition 1; then they *update* it by removing the part dominated by the optimal solution when it exists or by removing the part that is explored when no feasible solution exists. These two steps are repeated until the search region does not contain feasible points anymore.

Our algorithm uses the characterization formalized by Klamroth et al. (2015), which describes the search region using p-dimensional subregions called *search zones*, each zone being described using *local upper bounds* on each objective. Therefore, the search region can be defined as a union of zones:

$$S(N) = \bigcup_{u \in U(N)} z(u), \tag{3}$$

where $U(N)$ denotes the set of the upper bounds delimiting the search region induced by N.

When a new nondominated point $y \in S(N) \cap Y_{\text{ND}}$ is found, it is necessary to update the search region by removing the points dominated by y. For this purpose, each search zone $z(u) \subseteq S(N)$ containing y—such that $y \prec u$—must be subdivided into p new search zones described by the p following local upper bound: $u^1 = (y_1, u_2, \ldots, u_p)$, $u^2 = (u_1, y_2, u_3, \ldots, u_p)$, and $\ldots$, $u^p = (u_1, \ldots, u_{p-1}, y_p)$. Bound u^k is called the k^{th} child of u, $k \in \{1, \ldots, p\}$.

2.2.2. Properties. Updating the search region can, however, lead to redundancies by creating search zones that are contained in others. Formally, given two upper bounds u and u' in $U(N)$, u' or its associated search zone $z(u')$ is *redundant* if $u' \leqq u$. Indeed, only maximal search zones are needed to characterize $S(N)$ in (3). Therefore, we assume in the following that $U(N)$ contains only maximal local upper bounds. The following result, initially introduced in Klamroth et al. (2015, proposition 4.1), presents an interesting relation between N and $U(N)$, leading to the concept of *defining points*.

Proposition 2. *Bound u belongs to $U(N)$ if and only if, for any of its bounded component $u_k \neq M$, there exists $y \in N$ such that $y_k = u_k$ and $y_{-k} \prec u_{-k}$.*

Proof. Let $u \in U(N)$ be a bound such that $u_k \neq M$ and $\hat{u} = (u_1, \ldots, u_k + \varepsilon, \ldots, u_p)$ for any $\varepsilon > 0$. Because u is maximal, this is equivalent to saying that $z(\hat{u})$ contains a point $\hat{y}$ that is dominated by a point $y \in N$, whereas $z(u)$ does not. Thus, we have $y_{-k} \leq \hat{y}_{-k} \prec u_{-k}$ and $u_k \leq y_k \leq \hat{y}_k < u_k + \varepsilon$. Considering that this observation is true for any $\varepsilon > 0$, we get $y_k = u_k$. Hence, there exists a point $y \in N$, with $y_k = u_k$ and $y_{-k} \prec u_{-k}$. □

Definition 3. Let $u \in U(N)$. Points $y \in N$ satisfying Proposition 2 on a given objective k such that

$$\begin{cases} y_k = u_k, \\ y_{-k} \prec u_{-k}, \end{cases} \tag{4}$$

are referred to as the k^{th} *defining points* of bound u.

Because several points may define the same component of a bound, we denote by $N_k(u)$ the set of k^{th} defining points of bound u in N.

Using this result, Klamroth et al. (2015) propose an algorithm to update the search region efficiently. For each bound u, it is enough to keep track of the defining points of each bounded component. Then, when we consider a new nondominated point $y \prec u$, before creating u^l, the l^{th} child of u, we check whether at least one defining point of u still satisfies (4) for each bounded component k, $k \neq l$. Otherwise, u^l is redundant and can be discarded. See Algorithm 1 for implementation details.

3. A New Enumeration Algorithm

We propose a new enumeration algorithm that consists of iteratively exploring the search zones until all of these are proved to be empty, as in the generic algorithmic scheme of Klamroth et al. (2015). As in any enumeration algorithm, most of the computation time is spent solving the integer programs that explore the search zones. For this reason, we focus here on reducing the number and difficulty of the integer programs to be solved.

Three critical steps must be designed carefully so as to produce an efficient algorithm:

- exploration of a search zone
- update of the search region
- selection of the search zone to be explored

Obviously, these three steps are interrelated. The following three subsections describe in details each of these steps.

Algorithm 1 (Updating the search region using defining points as in Klamroth et al. (2015))

Input : $U(N)$: set of local upper bounds.
y^*: a new nondominated point.
Output: $U(N \cup \{y^*\})$: updated set of local upper bounds.

$U(N \cup \{y^*\}) \leftarrow U(N)$
foreach $u \in U(N)$ **do**
  if $y^* \prec u$ **then**
    $U(N \cup \{y^*\}) \leftarrow U(N \cup \{y^*\}) \setminus \{u\}$
    foreach $\ell \in \{1, \dots, p\}$ **do**
      /* Before computing the ℓ^{th} child, check if at least one defining point is still valid for each bounded component. */
      $u^\ell \leftarrow (u_1, \dots, u_{\ell-1}, y^*_\ell, u_{\ell+1}, \dots, u_p)$
      foreach $k \in \{1, \dots, p\},\ k \neq \ell$ **do**
        $N_k(u^\ell) \leftarrow \{y \in N_k(u), y_\ell < y^*_\ell\}$
      /* y^* obviously defines component ℓ of child u^ℓ */
      $N_\ell(u^\ell) \leftarrow \{y^*\}$
      if $N_k(u^\ell) \neq \emptyset$ *for all* $k \in \{1, \dots, p\} \setminus \{\ell\}$ *such that* $u^\ell_k \neq M$ **then**
        /* At least one defining point is still valid for each bounded component, u^ℓ is then maximal and must be added to the new search region */
        $U(N \cup \{y^*\}) \leftarrow U(N \cup \{y^*\}) \cup \{u^\ell\}$
  else if $y^*_k = u_k$ *and* $y^*_{-k} \prec u_k$ *for some* $k \in \{1, \dots, p\}$ **then**
    /* Update the k^{th} list of defining points for bound u */
    $N_k(u) \leftarrow N_k(u) \cup \{y^*\}$

3.1. Exploration Phase

Exploring a search zone $z(u)$ is performed using the following program, where objective k is minimized in priority and not constrained:

$$\Pi(k,u)\begin{cases} \text{lexmin} & \left\{ y_k, \sum_{\substack{i=1 \\ i \neq k}}^{p} y_i \right\} \\ \text{s.t.} & y \in Y, \\ & y_i < u_i \quad \forall i \in \{1, \dots, p\},\ i \neq k. \end{cases}$$

We first present some important properties regarding the feasibility and the optimal solution of problem $\Pi(k,u)$. Then, we show two ways of solving this program.

Proposition 3. *If problem $\Pi(k,u)$ is feasible, a nondominated point minimizing objective k over the zone delimited by the projected bound u_{-k} is returned.*

Proof. Direct consequence of Proposition 1. □

Observe that any optimal solution y^* of $\Pi(k,u)$ does not necessarily belong to the search region because the bound on objective k has been omitted. In other words, this means that y^* may have already been enumerated. Approaches relying only on projected bounds have to check this by browsing the entire list of enumerated points. However, because in our case the search region is precisely characterized, we can use the following results, which provide an efficient way to check whether y^* is a new nondominated point.

Proposition 4. *Let $u \in U(N)$, and let y^* be an optimal solution of program $\Pi(k,u)$. Then, we have $y^*_k \leq u_k$.*

Proof. If $u_k = M$, the result is trivial. Otherwise, u_k is bounded. Because $U(N)$ only contains maximal bounds, we have $N_k(u) \neq \emptyset$. Let $y \in N_k(u)$. From (4), we have $y_{-k} \prec u_{-k}$ and $y_k = u_k$. Therefore y is feasible for $\Pi(k,u)$, and we have $y^*_k \leq y_k = u_k$. □

We now clarify the condition under which y^* is a new nondominated point.

Corollary 1. *Let y^* be an optimal solution of program $\Pi(k,u)$. Point $y^* \in S(N)$ if and only if $y^* \notin N_k(u)$.*

Proof.
$\Rightarrow$: If $y^* \in S(N)$, by definition of the search region, we have $y^* \notin N$, and because $N_k(u) \subset N$, we have $y^* \notin N_k(u)$.
$\Leftarrow$: From Proposition 4, we have $y^*_k \leq u_k$. If $y^*_k < u_k$, then $y^* \prec u$; thus $y^* \in S(N)$. Otherwise, if $y^*_k = u_k$, y^* is a k^{th} defining point of bound u because it satisfies (4). Therefore, if $y^* \notin N_k(u)$, y^* is a new nondominated point. □

Contrary to the approaches that use projected bounds to characterize the search region, we can detect new nondominated points without checking the whole set N. In practice, if $\Pi(k,u)$ yields a point y^*, we check whether $y^*_k < u_k$. If so, y^* belongs to $z(u)$ and is thus a new nondominated point. Otherwise, we have $y^*_k = u_k$ with two possible subcases. Either y^* does not belong to $N_k(u)$, in which case y^* is also a new nondominated point that actually belongs to another zone than $z(u)$, or it belongs to $N_k(u)$ and thus has been already enumerated. Because $N_k(u)$ is usually a very small subset of N, checking whether y^* is new or not is performed very quickly in practice.

The following result, which also derives from Proposition 2, allows us to guarantee that $\Pi(k,u)$ is feasible when u_k is bounded. Moreover, it is possible to provide a feasible starting solution, which significantly decreases the time spent on the resolution.

Proposition 5. *Let $u \in U(N)$. If $u_k \neq M$, program $\Pi(k,u)$ is feasible, and there exists a point $y \in N \cap F(\Pi(k,u))$.*

Proof. From Proposition 2, because u is a maximal local upper bound such that $u_k \neq M$, $N_k(u)$ is nonempty. Then, any point $y \in N_k(u)$ satisfies, by definition, $y_{-k} < u_{-k}$ and is thus feasible for $\Pi(k,u)$. □

Therefore, keeping track of the sets of defining points $N_k(u)$ for each bound $u \in U(N)$ allows us to provide, in constant time, a feasible starting solution of program $\Pi(k,u)$ when $u_k \neq M$. Note that, except at the first iteration, it is always possible to select k such that u_k is bounded. It is then possible to guarantee that every subsequent program will be feasible.

The lexicographic program $\Pi(k,u)$ can be solved in two main ways: either by solving two consecutive programs (*two-stage approach*) or by solving a unique weighted sum problem (*direct approach*).

- *Two-Stage Approach.* We can obtain a nondominated point minimizing criterion k in the search zone $z(u)$ by solving consecutively two programs. First, we solve the following program:

$$\Pi^1(k,u)\begin{cases}\min & y_k\\ \text{s.t.} & y \in Y,\\ & y_i < u_i \quad \forall i \in \{1,\dots,p\},\ i \neq k,\end{cases}$$

which minimizes only objective k.

The optimal point y^* of $\Pi^1(k,u)$ is guaranteed only to be weakly nondominated because it could be dominated by another point reaching the same performance on the k^{th} criterion but better values on other criteria. It is then necessary to solve another program to obtain a nondominated point:

$$\Pi^2(k,u)\begin{cases}\min & \sum_{i=1}^{p} y_i\\ \text{s.t.} & y \in Y,\\ & y_k = y_k^*,\\ & y_i \le y_k^* \quad \forall i \in \{1,\dots,p\},\ i \neq k.\end{cases}$$

Because y^* is feasible for $\Pi^2(k,u)$, using it as a starting feasible solution can improve the resolution.

- *Direct Approach.* It is theoretically possible to merge both previous programs in order to obtain directly a nondominated point having the best performance on the k^{th} criterion. This can be done by solving a single program where the objective function involves all the remaining criteria: the main criterion has to be weighted using a sufficiently large coefficient in order to ensure that it is impossible to compensate any of its degradation with good values on the others. The following result provides a bound on this weight when the objective functions take integer values.

Proposition 6. *Assuming that all objective functions take integer values, the following program leads to a nondominated point reaching the lowest performance on the k^{th} criterion over zone $z(u_{-k})$, if feasible:*

$$\Pi'(k,u)\begin{cases}\min & \Delta y_k + \sum_{\substack{i=1\\ i\neq k}}^{p} y_i\\ \text{s.t.} & y \in Y,\\ & y_i < u_i \quad \forall i \in \{1,\dots,p\},\ i \neq k,\end{cases}$$

with $\Delta > \sum_{\substack{i=1\\ i\neq k}}^{p} (U_i - L_i)$, where U_i and L_i are the respective upper and lower bounds of any value reached by criterion f_i over zone $z(u_{-k})$.

Proof. By Proposition 3, $\Pi'(k,u)$ provides a nondominated point. Let y_k^* be the optimal value on criterion k over the zone delimited by the projected bound u_{-k}. For a sufficiently large Δ value, problem $\Pi'(k,u)$ yields a solution with value y_k^* on objective k. We look for the smallest value of Δ for which this remains true. The worst possible situation is the existence of the two following feasible points belonging to $F(\Pi(k,u))$:

$$y^1 = \left(U_1,\dots,U_{k-1},y_k^*,U_{k+1},\dots,U_p\right),$$
$$y^2 = \left(L_1,\dots,L_{k-1},y_k^*+1,L_{k+1},\dots,L_p\right).$$

Then Δ must be selected such that y^1 is still chosen; that is, $\Delta y_k^* + \sum_{\substack{i=1\\ i\neq k}}^{p} U_i < \Delta(y_k^*+1) + \sum_{\substack{i=1\\ i\neq k}}^{p} L_i$, which yields $\Delta > \sum_{\substack{i=1\\ i\neq k}}^{p} (U_i - L_i)$. □

Because any search zone is defined using a local upper bound, we can adapt the previous result.

Corollary 2. *Program $\Pi'(k,u)$ with $\Delta = 1 + \sum_{\substack{i=1\\ i\neq k}}^{p} (u_i - y_i^I)$ provides a nondominated point in the zone delimited by the projected bound u_{-k} minimizing the k^{th} criterion, if feasible.*

Proof. By definition of $\Pi(k,u)$, we have $u_i > U_i$ and $y_i^I \le L_i$. The result follows. □

The direct approach, which solves only one program instead of two at each iteration, is potentially faster. However, if coefficient Δ on objective f_k is too large when compared with coefficient 1 on the other objectives, this only means that we just optimize objective f_k without considering the other objectives, because of the lack of numerical precision. Then, we are guaranteed to yield at least a weakly nondominated point, which might be dominated on the $p-1$ remaining objectives. Therefore, the final returned set might include a few weakly nondominated points that are dominated but will certainly contain all nondominated points. It should be noted, however, that the use of a local upper bound u in the definition of Δ, instead of a global upper bound U over the whole set of feasible points, limits this risk. Nevertheless, when using the direct approach, an

additional procedure is required to filter out the weakly nondominated points that are dominated. Observe, however, that it is only necessary to compare the optimal point of $\Pi'(k,u)$ with the points having the same performance on objective k instead of browsing the whole set to detect weak dominance, which drastically reduces the number of needed comparisons.

3.2. Reduction Rules

Even if $U(N)$ contains only maximal bounds, some of them can be discarded by proving that the corresponding search zones do not contain feasible points, without testing them by solving a program Π.

A first trivial case occurs when a bound u reaches the performance of the ideal point on at least one objective (i.e., when $u_i = y^I_i$ for some $i \in \{1,\dots,p\}$). Then, search zone $z(u)$ cannot contain any feasible point and can be discarded because, by definition, no point can strictly improve the ideal point on any objective.

A second simple case is outlined in the next result, showing that one child can be discarded each time a new point is generated.

Proposition 7. *Let $u \in U(N)$, and let y^* be an optimal point of $\Pi(k,u)$ with $y^* \prec u$. Then, u^k, the k^{th} child of u, can be discarded.*

Proof. Because $u^k = (u_1,\dots,u_{k-1},y^*_k,u_{k+1},\dots,u_p)$, we have $F(\Pi(k,u^k)) = F(\Pi(k,u))$. Therefore, y^*_k is also the optimal value of $\Pi(k,u^k)$. It follows that $z(u^k)$ does not contain any feasible point. □

These first two cases are very simple to check. Proposition 7 can be generalized in order to detect other empty zones. This generalization will allow us to discard some zones by checking the results of programs Π previously solved. Owing to some specific properties, we show that we do not need to check all previously solved programs but only a small fraction of them.

Proposition 8. *Let $u \in U(N)$, and let y^*_k be the optimal value of $\Pi(k,u)$. Then, any bound u' such that*

$$\begin{cases} u'_{-k} \leqq u_{-k}, \\ u'_k \leq y^*_k, \end{cases} \tag{5}$$

can be discarded.

Proof. Because $F(\Pi(k,u')) \subset F(\Pi(k,u))$, y^*_k is a lower bound on objective k for any feasible point in $F(\Pi(k,u'))$. Then, if $u'_k \leq y^*_k$, zone $z(u')$ does not contain any feasible point. □

Proposition 7 is clearly a particular case of Proposition 8, with $u' = u^k$. Observe that, by construction, bound u^k is generated *after* bound u—that is, when $u \in U(N)$, $u^k \notin U(N)$. Moreover, instead of having $u'_k \leq y^*_k$, we have more particularly, by construction, $u^k_k = y^*_k$. We show that these two properties are actually also true for any bound u' that satisfies (5).

Proposition 9. *Let $u \in U(N)$, and let y^*_k be the optimal value of $\Pi(k,u)$. Any maximal bound u' that satisfies (5) is such that $u' \notin U(N)$ and $u'_k = y^*_k$.*

Proof. Using (5) and Proposition 4 we get $u'_k \leq y^*_k \leq u_k$. Because $u'_{-k} \leqq u_{-k}$, we have $u' \leqq u$. Owing to the maximality of bounds in $U(N)$, we have $u' \notin U(N)$, and it can thus appear at a subsequent iteration, which proves the first part of the result.

Let $y \in N_k(u')$ be a k^{th} defining point of u'. From (4), we have $y \in F(\Pi(k,u'))$ and $y_k = u'_k$. Because $F(\Pi(k,u')) \subset F(\Pi(k,u))$, we have $y^*_k \leq y_k = u'_k$. Therefore, if u' satisfies (5), we have $u'_k = y^*_k$. □

Combining Propositions 8 and 9, we deduce that we can discard a bound u' if we already solved (at a previous iteration) a program $\Pi(k,u)$ with optimal value y^* such that $u'_{-k} \leqq u_{-k}$ and $u'_k = y^*_k$. For this purpose, we maintain p lists $V(k)$, $k \in \{1,\dots,p\}$, each one keeping track of the already solved feasible programs $\Pi(k,u)$ with optimal value y^*_k, where each program is represented by the pair (u, y^*_k). These lists are sorted according to values y^*_k. Detecting the existence of a pair such that $y^*_k = u'_k$ (and $u'_{-k} \leqq u_{-k}$) can be performed efficiently using a dichotomic search, instead of browsing the whole list.

3.3. Selection of Search Zones

An important issue is the selection of the search zone $z(u)$ to be explored as well as the criterion k to be optimized over this search zone. This defines the program $\Pi(k,u)$ to be solved at the next iteration. We now describe a heuristic to guide the selection of a search zone.

First of all, as outlined in Section 3.1, we want to select k such that u_k is bounded because this allows us both to guarantee that program $\Pi(k,u)$ is feasible and to provide a feasible starting solution in constant time.

The heuristic aims at satisfying two other natural goals. First, we would like to favor zones that are more liable to contain a new nondominated point. Because we do not have any prior information, it seems reasonable to favor zones delimiting the largest unexplored part of the objective space. Second, we would like to select zones that have a larger chance to trigger the application of the reduction rules presented in Section 3.2 (i.e., zones with the largest projection).

In order to perform the selection of both the local upper bound and the objective to be optimized, we

propose the following priority index, which might be seen as the hypervolume of the projection u_{-k}:

$$h(k,u) = \prod_{i \neq k} \left(u_i - y_i^I\right),$$

where y^I is the global ideal point. At each iteration, bound u^* and the objective k^* are selected according to (6) in order to maximize the hypervolume:

$$\left(k^*, u^*\right) = \begin{cases} (1, (M, \ldots, M)) & \text{if } N = \emptyset, \\ \arg\max\limits_{\substack{u \in U(N) \\ k \in \{1,\ldots,p\} \\ u_k \neq M}} \{h(k,u)\} & \text{otherwise.} \end{cases} \tag{6}$$

Note that except at the first iteration, all of the search zones have at least one bounded component, allowing us to compute the index h.

The two following results support the choice of our heuristic index.

Proposition 10. *Except at the first iteration, program* $\Pi(k^*, u^*)$ *is necessarily feasible.*

Proof. Except at the first iteration, we have $u_{k^*} \neq M$ by construction. Then, the result is a direct consequence of Proposition 5. □

The following result proves that the projection of bound u^* according to objective k^* is not included in any other projection in the same direction.

Proposition 11. *There is no bound* $u \in U(N)$ *such that* $u^*_{-k^*} \leq u_{-k^*}$.

Proof. Suppose there exists $u \in U(N)$ such that $u^*_{-k^*} \leq u_{-k^*}$. This implies that $u^*_i \leq u_i,\ i \in \{1, \ldots, p\} \setminus \{k\}$, with $u^*_i < u_i$ for some i, leading to $h(k^*, u^*) < h(k^*, u)$, contradicting (6). □

Therefore, the projection of the bound with the largest hypervolume will not be included in another one, increasing the chance to trigger the reduction rule presented in the previous section.

Our algorithm including the refinements presented in the three previous subsections is described as Algorithm 2.

Algorithm 2 (Determination of the set of nondominated points)

Input: Y, image of the feasible set X
Output: Y_{ND}, the nondominated set of Y

```
/* Initialize the set of nondominated points, the list of upper bounds, and the set of solved models */
1 N ← ∅   U(N) ← {(M, ..., M)}   V(k) ← ∅, k ∈ {1, ..., p}
2 compute the ideal point y^I
3 while U(N) ≠ ∅ do
    /* Select the search zone and the criterion maximizing the score */
4   select (k*, u*) according to (6)
5   y* ← solve Π(k*, u*) /* using any y ∈ N_k*(u*) as a starting solution */
    /* Add the model to the archive of solved models */
6   V(k*) ← V(k*) ∪ {(u*, y*_k)}
7   if y* ∉ N_k*(u*) then
      /* y* is a new nondominated point. The search region is updated as described in Algorithm 1 */
8     U(N ∪ {y*}) ← updateSearchRegion(U(N), y*)
9     N ← N ∪ {y*}
    /* Clean the search region using the reduction rules */
10  foreach u' ∈ U(N) do
11    foreach k ∈ {1, ..., p} do
12      if u'_k = y^I_k then
13        U(N) ← U(N) \ {u'}
14      else
15        foreach (u, y_k) ∈ V(k) do
16          if u'_{-k} ≦ u_{-k} and y_k = u'_k then
17            U(N) ← U(N) \ {u'}
  /* All the nondominated points are in N. */
18 Y_ND ← N
```

4. Computational Results

Our algorithm has been implemented using the Haskell programming language, which calls the IBM ILOG CPLEX Concert Technology 12.9™ on an Intel i9 9900K 4.7 GHz CPU with 7.8 GB of memory. Because Kirlik and Sayin (2014) have already shown that their approach outperformed the other state-of-the-art algorithms at this date, we compare our algorithm to theirs, referred to as KS2014. We also compare our algorithm to the recent approach proposed by Boland et al. (2017), referred to as BCS2017. We used, on the same computer, the implementations provided by the authors (at http://home.ku.edu.tr/~moolibrary/ and https://ogma.newcastle.edu.au/vital/access/manager/Repository/uon:17056, respectively). Comparisons have been made on the multiobjective knapsack problem (MOKP) and multiobjective assignment problem (MOAP), which are described in the next section. Our implementation and the set of instances can be found at https://www.lamsade.dauphine.fr/~vdp/tv19.

4.1. Instances

4.1.1. Multiobjective Knapsack Problem. Consider a set of n items. Each item i has p values $v_i^k, k \in \{1, \ldots, p\}$, and a weight $\omega_i,\ i \in \{1, \ldots, n\}$. Given a capacity ω, we

Table 1. Comparison of the Different Approaches on MOKP Instances

p	n	Name	$\lvert Y_{ND}\rvert$ (mean)	*CPU* (mean)	*#Iterations* (mean)	*#Infeasible* (mean)	$\lvert U\rvert_{max}$ (mean)	$\lvert U\rvert_{avg}$ (mean)
2	200	BCS2017	405.4	36.683	716.8	311.4		
		KS2014	405.4	13.461	405.4	0.0	2.0	1.997
		Direct	405.4	9.336	406.4	0.0	1.0	1.000
		TwoStage	405.4	12.292	406.4	0.0	1.0	1.000
3	100	BCS2017	4,090.0	1,206.036	9,324.9	5,234.9		
		KS2014	4,090.0	427.681	7,306.5	125.4	110,919.9	91,571.580
		Direct	4,090.0	227.261	7,358.8	0.0	403.9	198.971
		TwoStage	4,090.0	273.524	7,446.9	0.0	377.9	190.969
4	50	BCS2017	2,240.1	816.821	9,734.0	7,493.9		
		KS2014	—	—	—	—	—	—
		Direct	2,240.1	184.471	10,280.7	0.0	3,396.6	1,828.558
		TwoStage	2,240.1	210.661	10,333.2	0.0	3,348.4	1,806.205
5	50	BCS2017	—	—	—	—	—	—
		KS2014	—	—	—	—	—	—
		Direct	8,788.3	4,221.594	121,795.4	0.0	54,501.2	32,833.959
		TwoStage	8,788.3	4,298.001	122,051.8	0.0	54,101.0	32,540.438
6	30	BCS2017	—	—	—	—	—	—
		KS2014	—	—	—	—	—	—
		Direct	1,710.4	1,434.566	61,075.1	0.0	39,030.9	23,450.765
		TwoStage	1,710.4	1,380.299	61,103.0	0.0	38,909.5	23,395.098

want to select a subset of items that maximizes the values of the knapsack while respecting capacity ω:

$$(\text{MOKP})\begin{cases}\max & \sum_{i=1}^{n} v_i^k x_i & k \in \{1,\dots,p\}\\ \text{s.t.} & \sum_{i=1}^{n} \omega_i x_i \le \omega, & \\ & x_i \in \{0,1\} & i \in \{1,\dots,n\}.\end{cases}$$

All values and weights are randomly generated in the interval $[1,100]$, and we set $\omega = \lceil \frac{\sum_{i=1}^{n} v_i}{2} \rceil$.

4.1.2. Multiobjective Assignment Problem. Consider a set I of agents and a set J of tasks with $|I| = |J| = n$. Each pair $(i,j) \in I \times J$ is valued by p costs c_{ij}^k to be minimized, $k \in \{1,\dots,p\}$. The goal is to assign each agent to exactly one machine:

$$(\text{MOAP})\begin{cases}\min & \sum_{i=1}^{n} c_{ij}^k x_{ij} & k \in \{1,\dots,p\}\\ \text{s.t.} & \sum_{i=1}^{n} x_{ij} = 1 & j \in \{1,\dots,n\},\\ & \sum_{j=1}^{n} x_{ij} = 1 & i \in \{1,\dots,n\},\\ & x_{ij} \in \{0,1\} & (i,j) \in \{1,\dots,n\}^2.\end{cases}$$

Each cost is randomly generated in the interval $[1,15]$.

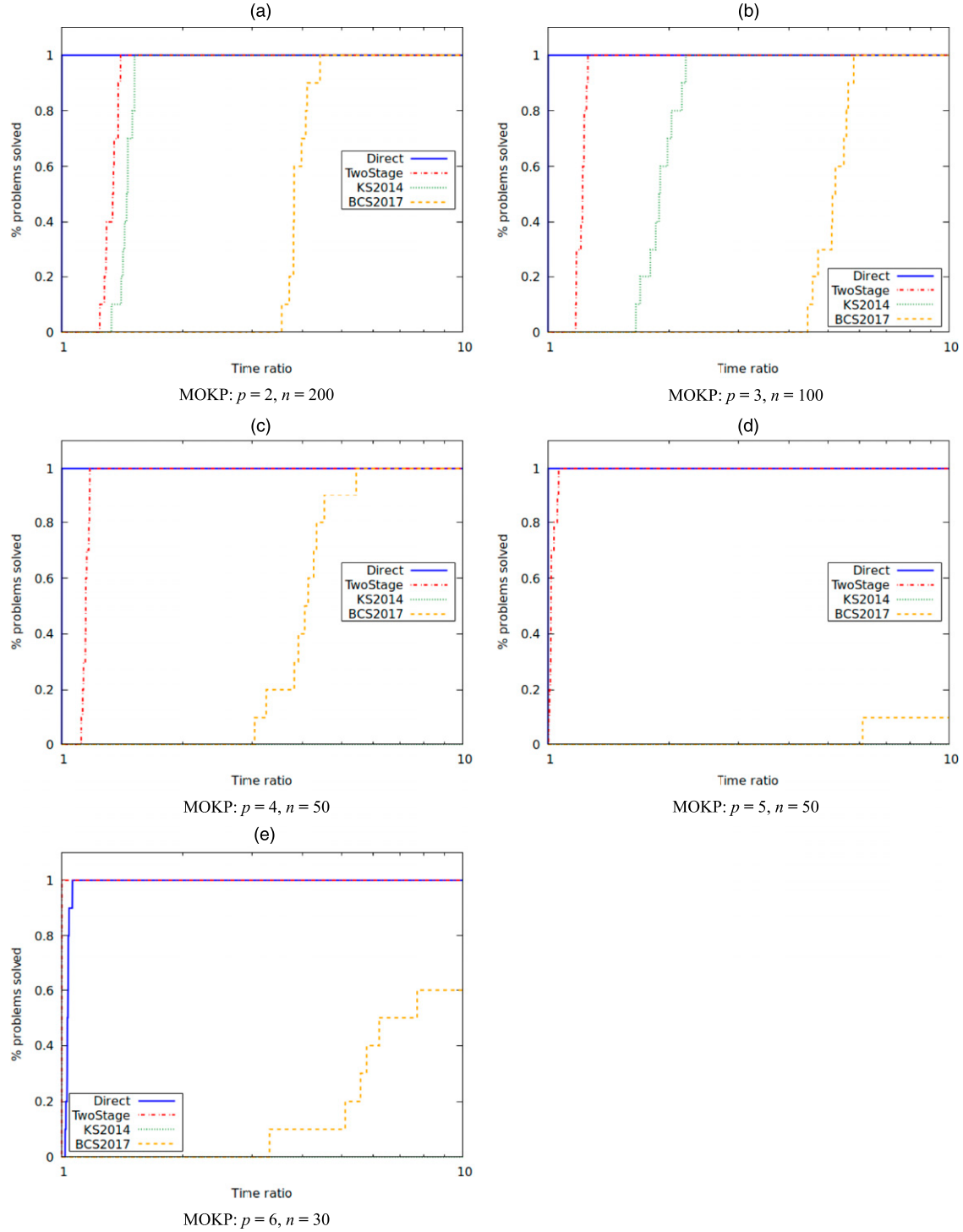
4.2. Analysis

For each type of instance, comparisons have been performed on a set of 10 instances. Results are reported in tables that indicate the average size of the final set ($|Y_{ND}|$), the computation times (*CPU*, in seconds), the number of iterations (*#Iterations*), the

Table 2. Comparison of the Different Approaches on MOAP Instances

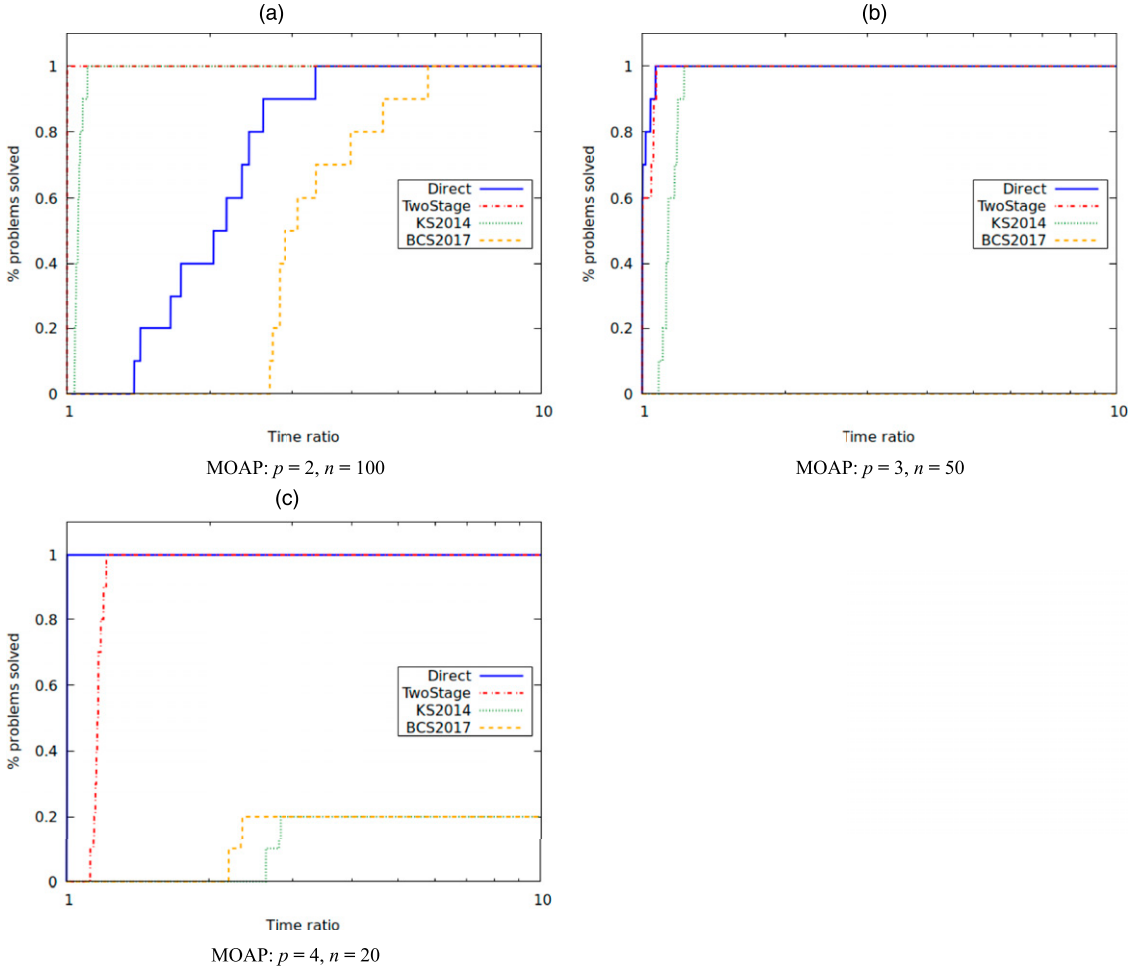
p	$n \times n$	Name	$\lvert Y_{ND}\rvert$ (mean)	*CPU* (mean)	*#Iterations* (mean)	*#Infeasible* (mean)	$\lvert U\rvert_{max}$ (mean)	$\lvert U\rvert_{avg}$ (mean)
2	100×100	BCS2017	202.8	217.41	391.2	188.4		
		KS2014	202.8	66.23	202.8	0.0	2.0	1.995
		Direct	202.8	131.64	203.8	0.0	1.0	1.000
		TwoStage	202.8	62.54	203.8	0.0	1.0	1.000
3	50×50	BCS2017	—	—	—	—	—	—
		KS2014	16,867.2	4,756.93	20,558.7	114.7	39,924.0	38,896.590
		Direct	16,867.2	4,212.61	23,538.1	0.0	524.3	318.472
		TwoStage	16,867.2	4,246.24	25,305.2	0.0	375.9	262.773
4	20×20	BCS2017	—	—	—	—	—	—
		KS2014	—	—	—	—	—	—
		Direct	24,750.0	4,134.80	62,027.1	0.0	9,331.4	5,510.141
		TwoStage	24,750.0	4,789.83	64,406.3	0.0	7,999.6	4,897.204

Figure 1. (Color online) Performance Profiles on MOKP (Logarithmic Scales), Where Higher Is Better



number of infeasible models (*#Infeasible*), and the maximal and average number of elements describing the search region ($|U|_{\max}$ and $|U|_{\text{avg}}$, respectively)—that is, the number of local upper bounds for our approaches and the number of boxes for KS2014. Because the search region of BCS2017 involves more complex objects, we omit these two last fields for this method. Results on MOKP and MOAP instances are synthesized in Tables 1 and 2, respectively. If at least one instance cannot be solved (either if the memory is saturated or if it is still unsolved after two hours), we report that the considered algorithm fails to solve the set. As a consequence, CPU time and the other measures are not computed when at least one instance

Figure 2. (Color online) Performance Profiles on MOAP (Logarithmic Scales), Where Higher Is Better



is not solved. We also provide *performance profiles* that describe the percentage of problems solved by each algorithm in terms of the time ratio relative to the fastest algorithm (Dolan and Moré 2002). Besides showing the algorithms' performance in relative terms, these profiles show the number of instances solved for each type of instance. Performance profiles for MOKP and MOAP instances are presented in Figures 1 and 2, respectively.

We first observe that the direct approach is generally faster than the two-stage approach, as expected. However, as pointed out in Section 3.1, using the direct approach may enumerate additional weakly nondominated points for numerical reasons. We observe, however, that this never occurs in our experiments, even when the size of the nondominated set is huge (see column "$|Y_{ND}|$," which is always identical for the two variants). Indeed, in practice, the largest weight on the optimized objective remains relatively small.

Comparisons with two state-of-the-art algorithms are also presented. Our experiments show that our algorithms outperform them. Both are clearly slower to compute the final set, which is probably due to the infeasible programs they need to solve and our ability to provide a feasible starting solution at each iteration.

Even if KS2014 needs slightly fewer iterations than our algorithm to compute the final set, KS2014 is clearly slower because of the huge amount of boxes, particularly for problems with more than three objectives: for example, in MOKP with three objectives and 100 items, both variants of our algorithm store at most 400 upper bounds in $U(N)$, whereas KS2014 requires storing more than 110,000 boxes. Thus, much more time is needed to update the search region. Moreover, the list of boxes saturates the memory when the number of objective grows, and KS2014 is not able to solve instances with four or more objectives, even when the size of the final set is small (see MOKP instances with six objectives and 30 items where $|Y_{ND}|$ is only about 1,710 in average). Figure 1 shows that KS2014 does not solve any instance of MOKP with more than three objectives, and Figure 2 shows that only two instances of MOAP with four objectives are solved. In conclusion, KS2014 seems

to be faster than BCS2017 for instances with at most three objectives, but it is outperformed for more than three objectives owing to the huge number of boxes, which causes either prohibitive computation times (see, e.g., MOAP instances with four objectives and 20 items in Figure 2) or memory issues.

By contrast, BCS2017 is more stable than KS2014 in the sense that it is not affected by memory issues. The algorithm requires, however, relatively large computation times: see MOKP and MOAP instances with two or three objectives, where this algorithm is clearly outperformed. This can be explained both by the fact that each integer program is harder to solve because of the disjunctive constraints and by the large number of infeasible programs among these (see, e.g., column "*#Infeasible*" in Table 1). Note that for MOAP instances with four objectives, BCS2017 outperforms KS2014 as a result of the very large number of boxes to be handled by KS2014. Finally, when the number of objectives and the size of the nondominated set grow (see, e.g., MOKP instances with six objectives and 30 items), both KS2014 and BCS2017 fail but for different reasons: there is an explosion in the number of boxes for KS2014 and huge computation times for BCS2017.

All these algorithms were specifically designed for multiobjective problems with at least three objectives. It is interesting, however, to analyse their behavior in the biobjective case. Our approach presents a lot of similarities with the classic ε-constraint method. We need exactly $|Y_{ND}| + 1$ iterations to compute the final set, and the search region contains exactly one search zone at each iteration. The only difference lies in the choice of the objective to be optimized: in the ε-constraint approach, the optimized objective is always the same; however, it may alternate in our approach, allowing us to provide a feasible starting solution as described previously. From these observations, we can see our approach as a generalization of the traditional ε-constraint method. KS2014 can also be interpreted as an extension of the ε-constraint method and is even more similar to this method because the optimized objective is always the same. As a consequence, both KS2014 and our approaches perform similarly in the biobjective case. BCS2017, which cannot be interpreted as an extension of the ε-constraint method, is slower in the biobjective case.

5. Conclusions and Perspectives

We present in this paper a new algorithm for computing the set of nondominated points. Our comparisons with two state-of-the-art algorithms show significantly better results on the experimented problem instances. Its main distinguishing features are (i) a guarantee that no infeasible program has to be solved during the exploration, (ii) that each program will be provided with a starting feasible solution (warm start), and (iii) an exact characterization of the search region using full-dimensional search zones. As a consequence, this algorithm benefits from the use of both the extra knowledge given by exploring the projection of each search region and the exact characterization of the search region using full-dimensional search zones, which provides stability when the number of objectives grows. Further works could concern the adaptation of this procedure to other related questions, such as the computation of the nadir point or the approximation of the nondominated points with an a priori guarantee on the quality of the returned set.

Acknowledgments

The authors are grateful to Kathrin Klamroth and Kerstin Dächert for their heplful discussions and comments.

Appendix. Comprehensive Running Example

Consider the following triobjective knapsack instance (corresponding to instance KP_p-3_n-10_ins-1.dat provided with the implementation of Kirlik and Sayin (2014)):

$$\begin{cases} \min\ 5{,}000 & -566x_1 - 611x_2 - 506x_3 - 180x_4 - 817x_5 \\ & -184x_6 - 585x_7 - 423x_8 - 26x_9 - 317x_{10} \\ \min\ 5{,}000 & -62x_1 - 84x_2 - 977x_3 - 979x_4 - 874x_5 \\ & -54x_6 - 269x_7 - 93x_8 - 881x_9 - 563x_{10} \\ \min\ 5{,}000 & -664x_1 - 982x_2 - 962x_3 - 140x_4 - 224x_5 \\ & -215x_6 - 12x_7 - 869x_8 - 332x_9 - 537x_{10} \\ \text{s.t.} & 557x_1 + 898x_2 + 148x_3 + 63x_4 + 78x_5 \\ & +964x_6 + 246x_7 + 662x_8 + 386x_9 \\ & +272x_{10} \le 2{,}137, \\ & x_i \in \{0,1\},\ i \in \{1,\dots,10\}. \end{cases}$$

For each iteration, we specify the current search region. The selected zone u^* is in bold, and the objective k^* to be optimized in $\Pi(k^*, u^*)$ is underlined.

1. Current search region: **[<u>M</u>, M, M]**
   - New point: [1,606, 1,183, 1,592]
   - New local upper bounds: [M, 1,183, M] [M, M, 1,592]
2. Current search region: [M, 1183, M] **[M, M, <u>1,592</u>]**
   - Starting from [1,606, 1,183, 1,592]
   - New point: [2,146, 1,430, 1,286]
   - New local upper bounds: [2,146, M, 1,592] [M, 1,430, 1,592]
3. Current search region: **[M, <u>1,183</u>, M]** [2,146, M, 1,592] [M, 1,430, 1,592]
   - Starting from [1,606, 1,183, 1,592]
   - New point: [2,146, 364, 1,924]
   - New local upper bounds: [2,146, 1,183, M] [M, 1,183, 1,924]
4. Current search region: [2,146, M, 1,592] **[M, 1,430, <u>1,592</u>]** [2,146, 1,183, M] [M, 1,183, 1,924]
   - Starting from [1,606, 1,183, 1,592]
   - Point [1,606, 1,183, 1,592] is dominated by defining point [1,606, 1,183, 1,592]

5. Current search region: **[2,146, 1,183, M]** [M, 1,183, 1,924] [2,146, M, 1,592]
   - Starting from [2,146, 364, 1,924]
   - New point: [1,958, 373, 1,811]
   - New local upper bounds: [1,958, 1,183, M] [2,146, 373, M] [M, 373, 1,924] [M, 1,183, 1,811]
   - Reduction rule triggered on zones: [1,958, 1,183, M]

6. Current search region: [2,146, M, 1,592] [2,146, 373, M] [M, 373, 1,924] **[M, 1,183, 1,811]**
   - Starting from [1,958, 373, 1,811]
   - New point: [2,294, 1,143, 1,696]
   - New local upper bounds: [2,294, 1,183, 1,811] [M, 1,143, 1,811] [M, 1,183, 1,696]
   - Reduction rule triggered on zones: [M, 1,183, 1,696]

7. Current search region: [2,146, 373, M] [M, 373, 1,924] [2,146, M, 1,592] [2,294, 1,183, 1,811] **[M, 1,143, 1,811]**
   - Starting from [1,958, 373, 1,811]
   - New point: [2,482, 1,134, 1,809]
   - New local upper bounds: [2,482, 1,143, 1,811] [M, 1,134, 1,811] [M, 1,143, 1,809]
   - Reduction rule triggered on zones: [M, 1,143, 1,809]

8. Current search region: [M, 373, 1,924] [2,146, M, 1,592] [2,294, 1,183, 1,811] [2,146, 373, M] [2,482, 1,143, 1,811] **[M, 1,134, 1,811]**
   - Starting from [1,958, 373, 1,811]
   - Point [1,958, 373, 1,811] is dominated by defining point [1,958, 373, 1,811]

9. Current search region: **[2,146, M, 1,592]** [2,294, 1,183, 1,811] [2,146, 373, M] [2,482, 1,143, 1,811] [M, 373, 1,924]
   - Starting from [1,606, 1,183, 1,592]
   - New point: [2,003, 1,461, 1,491]
   - New local upper bounds: [2,003, M, 1,592] [2,146, 1,461, 1,592] [2,146, M, 1,491]
   - Reduction rule triggered on zones: [2,146, M, 1,491]

10. Current search region: [2,294, 1,183, 1,811] [2,146, 373, M] [2,482, 1,143, 1,811] **[M, 373, 1,924]** [2,003, M, 1,592] [2,146, 1,461, 1,592]
   - Starting from [1,958, 373, 1,811]
   - Point [1,958, 373, 1,811] is dominated by defining point [1,958, 373, 1,811]

11. Current search region: **[2,146, 373, M]** [2,482, 1,143, 1,811] [2,003, M, 1,592] [2,146, 1,461, 1,592] [2,294, 1,183, 1,811]
   - Starting from [1,958, 373, 1,811]
   - Point [1,958, 373, 1,811] is dominated by defining point [1,958, 373, 1,811]

12. Current search region: [2,482, 1,143, 1,811] **[2,003, M, 1,592]** [2,146, 1,461, 1,592] [2,294, 1,183, 1,811]
   - Starting from [1,606, 1,183, 1,592]
   - Point [1,606, 1,183, 1,592] is dominated by defining point [1,606, 1,183, 1,592]

13. Current search region: **[2,482, 1,143, 1,811]** [2,146, 1,461, 1,592] [2,294, 1,183, 1,811]
   - Starting from [1,958, 373, 1,811]
   - Point [1,958, 373, 1,811] is dominated by defining point [1,958, 373, 1,811]

14. Current search region: **[2,146, 1,461, 1,592]** [2,294, 1,183, 1,811]
   - Starting from [1,606, 1,183, 1,592]
   - Point [1,606, 1,183, 1,592] is dominated by defining point [1,606, 1,183, 1,592]

15. Current search region: **[2,294, 1,183, 1,811]**
   - Starting from [1,958, 373, 1,811]
   - Point [1,958, 373, 1,811] is dominated by defining point [1,958, 373, 1,811]

References

Bazgan C, Hugot H, Vanderpooten D (2009) Solving efficiently the 0–1 multi-objective knapsack problem. *Comput. Oper. Res.* 36(1): 260–279.

Boland N, Charkhgard H, Savelsbergh M (2017) A new method for optimizing a linear function over the efficient set of a multiobjective integer program. *Eur. J. Oper. Res.* 260(3):904–919.

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

Chankong V, Haimes YY (1983) *Multiobjective Decision Making: Theory and Methodology* (North Holland, New York).

Dächert K, Klamroth K, Lacour R, Vanderpooten D (2017) Efficient computation of the search region in multi-objective optimization. *Eur. J. Oper. Res.* 260(3):841–855.

Dhaenens C, Lemesre J, Talbi E (2010) *K*-PPM: A new exact method to solve multi-objective combinatorial optimization problems. *Eur. J. Oper. Res.* 200(1):45–53.

Dolan ED, Moré JJ (2002) Benchmarking optimization software with performance profiles. *Math. Programming* 91(2):201–213.

Kirlik G, Sayin S (2014) A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems. *Eur. J. Oper. Res.* 232(3):479–488.

Klamroth K, Lacour R, Vanderpooten D (2015) On the representation of the search region in multi-objective optimization. *Eur. J. Oper. Res.* 245(3):767–778.

Klein D, Hannan EL (1982) An algorithm for the multiple objective integer linear programming problem. *Eur. J. Oper. Res.* 9(4):378–385.

Laumanns M, Thiele L, Zitzler E (2005) An adaptive scheme to generate the Pareto front based on the epsilon-constraint method. Branke J, Deb K, Miettinen K, Steuer RE, eds. *Dagstuhl Seminar Proc.* (Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Saarbrücken, Germany).

Lokman B, Köksalan M (2013) Finding all nondominated points of multi-objective integer programs. *J. Global Optim.* 57(2):347–365.

Martins EQV (1984) On a multicriteria shortest path problem. *Eur. J. Oper. Res.* 16(2):236–245.

Mavrotas G (2009) Effective implementation of the ε-constraint method in multi-objective mathematical programming problems. *Appl. Math. Comput.* 213(2):455–465.

Mavrotas G, Florios K (2013) An improved version of the augmented ε-constraint method (augmecon2) for finding the exact Pareto set in multi-objective integer programming problems. *Appl. Math. Comput.* 219(18):9652–9669.

Özlen M, Azizoğlu M (2009) Multi-objective integer programming: A general approach for generating all non-dominated solutions. *Eur. J. Oper. Res.* 199(1):25–35.

Özlen M, Burton BA, MacRae CA (2014) Multi-objective integer programming: An improved recursive algorithm. *J. Optim. Theory Appl.* 160(2):470–482.

Przybylski A, Gandibleux X (2017) Multi-objective branch and bound. *Eur. J. Oper. Res.* 260(3):856–872.

Sylva J, Crema A (2004) A method for finding the set of non-dominated vectors for multiple objective integer linear programs. *Eur. J. Oper. Res.* 158(1):46–55.

Sylva J, Crema A (2008) Enumerating the set of non-dominated vectors in multiple objective integer linear programming. *RAIRO Oper. Res.* 42(3):371–387.

Zhang W, Reimann M (2014) A simple augmented ε-constraint method for multi-objective mathematical integer programming problems. *Eur. J. Oper. Res.* 234(1):15–24.