Optimizing over the Efficient Set of a Multi-Objective Discrete Optimization Problem

Satya Tamby, Daniel Vanderpooten

Optimizing over the Efficient Set of a Multi-Objective Discrete Optimization Problem

Satya Tamby
Université Paris Dauphine, PSL Research University, LAMSADE, France

Daniel Vanderpooten[1]
Université Paris Dauphine, PSL Research University, LAMSADE, France

Abstract

Optimizing over the efficient set of a discrete multi-objective problem is a challenging issue. The main reason is that, unlike when optimizing over the feasible set, the efficient set is implicitly characterized. Therefore, methods designed for this purpose iteratively generate efficient solutions by solving appropriate single-objective problems. However, the number of efficient solutions can be quite large and the problems to be solved can be difficult practically. Thus, the challenge is both to minimize the number of iterations and to reduce the difficulty of the problems to be solved at each iteration.

In this paper, a new enumeration scheme is proposed. By introducing some constraints and optimizing over projections of the search region, potentially large parts of the search space can be discarded, drastically reducing the number of iterations. Moreover, the single-objective programs to be solved can be guaranteed to be feasible, and a starting solution can be provided allowing warm start resolutions. This results in a fast algorithm that is simple to implement.

Experimental computations on two standard multi-objective instance families show that our approach seems to perform significantly faster than the state of the art algorithm.

2012 ACM Subject Classification Applied computing → Multi-criterion optimization and decision-making; Theory of computation → Integer programming

Keywords and phrases discrete optimization, multi-objective optimization, non-dominated set, efficient set

Digital Object Identifier 10.4230/LIPIcs.SEA.2023.9

1 Introduction

For problems involving multiple objectives, solutions of interest are *efficient* solutions for which there is no other solution which dominates it, meaning that it is at least as good on all objectives and strictly better on at least one objective. The resulting efficient set is often of large cardinality for multi-objective discrete problems, and in particular multi-objective combinatorial optimization (MOCO) problems. In order to discriminate among efficient solutions, a natural approach is to optimize, over the efficient set, a value function Φ which represents a major objective or the preferences of a specific decision maker. A special case of interest is the determination of the nadir point which, when considering objectives to be minimized, corresponds to the worst values achieved by efficient solutions for each objective. This valuable information allowing a decision maker to better appreciate the values that he/she could expect, can indeed be seen as maximizing independently each objective function over the efficient set.

When the function Φ to be optimized guarantees to return an efficient solution (e.g. when Φ is a positively weighted sum of the objective functions), optimizing Φ over the efficient set can be performed by optimizing Φ over the feasible set. In general, however, optimizing

[1] corresponding author

Φ directly over the feasible set will return a dominated solution. The difficulty stems from the fact that, unlike the feasible set, the efficient set is not explicitly defined by a set of constraints.

A trivial approach consists of enumerating the entire efficient set, computing the image of each solution through function Φ before finding the optimal one. However, as mentioned before, this is not a convenient approach since computing the efficient set can be intractable due to its large cardinality. For this reason, most approaches, including ours, try to find an optimal solution by enumerating the smallest possible subset of efficient solutions.

After stating the problem formally in Section 2, we briefly review the literature indicating the positioning of our approach among existing approaches (Section 3). We then state some preliminary results (Section 4) before presenting our algorithm in Section 5. Experimental results on two standard MOCO problems are then reported in Section 6. Some conclusions and perspectives are finally presented.

2 Problem statement

In the following, vectors are written in bold contrarily to scalars. Components of vectors are specified as indices.

2.1 Basic definitions and notations

Given a discrete set $\mathcal{X}$ of feasible solutions, defined by constraints on n decision variables, and p objective functions or criteria $\boldsymbol{f} = (f_1, \ldots, f_p)$,

we consider the following multi-objective problem:

$$\text{(MOP)} \qquad \begin{cases} \min & \boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_p(\boldsymbol{x})) \\ \text{s.t.} & \boldsymbol{x} \in \mathcal{X} \end{cases}$$

For any feasible solution $\boldsymbol{x} \in \mathcal{X}$, its image $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$ is referred to as a *feasible point* and $\mathcal{Y} = \boldsymbol{f}(\mathcal{X})$ denotes the set of feasible points. In this setting $\mathbb{R}^n$ and $\mathbb{R}^p$, will be referred to as the *decision space* and the *objective space*, respectively.

Given p dimensional points in $\mathbb{R}^p$, we consider the following binary relations; they are respectively referred to as *(Pareto) dominance*, *strong dominance* and *weak dominance*:

$$\begin{aligned} \boldsymbol{y} \preceq \boldsymbol{y}' &\iff \begin{cases} \boldsymbol{y}_i \leq \boldsymbol{y}'_i & \forall i \in \{1, \ldots, p\} \\ \boldsymbol{y} \neq \boldsymbol{y}' \end{cases} \\ \boldsymbol{y} \prec \boldsymbol{y}' &\iff \boldsymbol{y}_i < \boldsymbol{y}'_i \quad \forall i \in \{1, \ldots, p\} \\ \boldsymbol{y} \leqq \boldsymbol{y}' &\iff \boldsymbol{y}_i \leq \boldsymbol{y}'_i \quad \forall i \in \{1, \ldots, p\} \end{aligned}$$

The set $\mathcal{Y}_N$, which contains the points that are non-dominated, is defined by: $\mathcal{Y}_N = \{\boldsymbol{y} \in \mathcal{Y}, \nexists \boldsymbol{y}' \in \mathcal{Y}, \boldsymbol{y}' \preceq \boldsymbol{y}\}$. The subset of feasible solutions that lead to a non-dominated point is referred to as *the efficient set* and is denoted by $\mathcal{X}_E = \boldsymbol{f}^{-1}(\mathcal{Y}_N)$. It should be observed that several efficient solutions may correspond to the same non-dominated point. Solving problem (MOP) is then usually understood as determining $\mathcal{Y}_N$ and providing one efficient solution associated with each non-dominated point in $\mathcal{Y}_N$. Many algorithms have been proposed for solving problem (MOP) in the discrete case including [9, 13, 12, 6, 2, 14]. As will be seen in Section 3, algorithms for optimizing over the efficient set have been strongly influenced by these algorithms.

Finally, $\boldsymbol{y}_{-k} \in \mathbb{R}^{p-1}$ denotes the *projection of* $\boldsymbol{y}$ *in the direction* k *i.e.* the point $\boldsymbol{y}$ where component k has been omitted, that is $\boldsymbol{y}_{-k} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{k-1}, \boldsymbol{y}_{k+1}, \ldots, \boldsymbol{y}_p)$.

2.2 Problem statement

Given a function $\Phi : \mathcal{X} \to \mathbb{R}$ to be minimized, the *problem of optimizing* Φ *over the efficient set of* X can be stated as follows:

$$(\text{MOP}_E) \qquad \begin{cases} \min & \Phi(\boldsymbol{x}) \\ \text{s.t.} & \boldsymbol{x} \in \mathcal{X}_E \end{cases}$$

The difficulty of this problem stems from the fact that $\mathcal{X}_E$ is not characterized explicitly, *i.e.* as a set of constraints. Note that under certain assumptions on Φ, this problem amounts to optimizing over the *feasible set*, making the problem much simpler. In particular, this is the case when the optima of Φ are guaranteed to be non-dominated, as stated later in Theorem 2. In general, however, optimizing over the feasible set returns a solution which is not efficient and provides a lower bound on Φ which may be very far from the optimal value.

3 Related works and contribution of this paper

Approaches optimizing over the efficient set usually rely on the concept of *search region* that has been formalized in [8, 4], which is described in Section 4.1. Informally, the search region associated to a set of points N corresponds to the subset of the objective space containing points not dominated by any point in N.

Most methods dealing with the discrete case for problem (MOP_E) follow the same general scheme. They iteratively minimize Φ over the current search region to obtain a candidate point $\boldsymbol{y}$. Since $\boldsymbol{y}$ is potentially dominated, an additional effort is required to check the Pareto-optimality of the candidate. This step is usually performed by solving a program leading to a point $\boldsymbol{y}' \in \mathcal{Y}_N$ that dominates $\boldsymbol{y}$. Among all efficient solutions $\boldsymbol{x}'$ corresponding to $\boldsymbol{y}'$, *i.e.* such that $\boldsymbol{x}' \in \boldsymbol{f}^{-1}(\boldsymbol{y}')$, one optimizing function Φ is selected and retained if it improves the current best value of Φ. Convergence is reached when there is no feasible point in the current search region or when the solution of the first phase is non-dominated.

The evolution of the proposed methods for optimizing over the efficient set of a discrete multi-objective problem (problem (MOP_E)) follows the evolution of the proposed methods for generating the non-dominated set of a discrete multi-objective problem (problem (MOP)) and is actually related to the evolution of the way of representing the search region.

The oldest methods for solving problem (MOP), such as [9, 13], used a complete and implicit representation of the search region. This involves imposing constraints stating that the new non-dominated point to be generated should improve on at least one objective with respect to *all* non-dominated points previously generated. This may be achieved by adding disjunctive constraints as shown in [13]. While quite easy to implement, the main drawback of this approach is the growth of the number of constraints which makes it impossible to solve other than small size instances. Similarly, the oldest methods for solving problem (MOP_E), such as [5, 3] use a complete and implicit representation of the search region, with the same drawbacks as for problem (MOP).

The current methods for solving problem (MOP) resort to a decomposition of the search region into a union of search zones which allows solving at each iteration problems of constant size testing the existence of new non-dominated points in a search zone. The clear advantage is that the required optimization is relatively fast. The corresponding algorithms, such as [12, 6, 2, 14], mostly differ on how *search zones are defined* (note that sometimes a superset of the search region is stored), how *search zones are explored* (*i.e.* which search zone should be explored first and how) and how the *search region is updated* (*i.e.* how to modify the search zones so as to remove the part dominated by a new point) - see [14] for more details.

Following this evolution, the most recent algorithms for solving problem (MOP_E) resort to a decomposition of the search region, and were often proposed by the same authors who adapted their decomposition approach to problem (MOP_E) [1, 11] or to its special case of determining the nadir point [10, 7]. Similarly the algorithm proposed in this paper can be seen as an adaptation of our previous algorithm for problem (MOP) presented in [14]. While preserving the positive features of our previous approach (the definition of rules allowing many search zones to be discarded without exploring them, the guarantee that the required optimization problems are feasible and the existence of an initial feasible solution provided to the solver, which considerably speeds up the solution times,...), our adaptation also includes new positive features specific to problem (MOP_E) . In particular, by focusing on the iterative improvement of function Φ, we define new rules to discard additional search zones which cannot contain efficient solutions improving Φ.

4 Preliminary results

4.1 Search region, search zones

Given a set of N points, the corresponding search region denoted by $S(N)$ corresponds to the set of points that are not dominated by a point of N, *i.e.*

$$S(N) = \{\boldsymbol{y} \in \mathbb{R}^p : \nexists \bar{\boldsymbol{y}} \in N, \bar{\boldsymbol{y}} \leqq \boldsymbol{y}\}$$

The search region, which describes the part of the objective space where undiscovered non-dominated points may lie, can be defined as a union of *search zones* delimited by *local upper bounds* (see [8] for more details). Denoting $U(N)$ as the set of these local upper bounds, we have then:

$$\boldsymbol{y} \in S(N) \iff \exists \mathbf{u} \in U(N) : \boldsymbol{y} \prec \mathbf{u}$$

When a new point $\boldsymbol{y}$ is found, the search region must be updated by removing the part dominated by $\boldsymbol{y}$. This is done by splitting each zone strictly dominated by $\boldsymbol{y}$ into p new zones, referred to as *children*. Child i of $\mathbf{u}$ is $\mathbf{u}^i = (\mathbf{u}_1, \ldots, \mathbf{u}_{i-1}, \boldsymbol{y}_i, \mathbf{u}_{i+1}, \ldots, \mathbf{u}_p)$.

Updating the search region can lead to redundancies, *i.e.* zones that are included in others. Since only maximal zones are required to represent the search region, [8, 4] have proposed some methods to avoid generating redundant zones. One of these relies on the identification of the *defining points*, that are the points in N which define the components of the local upper bounds.

▶ **Definition 1.** *A point $\boldsymbol{y} \in N$ is a* defining point *for the component i of $\mathbf{u}$ if and only if $\boldsymbol{y}_i = \mathbf{u}_i$ and $\boldsymbol{y}_{-i} \prec \mathbf{u}_{-i}$.*

The following result allows the efficient identification of maximal local upper bounds.

▶ **Theorem 1** ([8])**.** *$\mathbf{u}$ is maximal if there exists at least one defining point for every bounded component of $\mathbf{u}$.*

In the following, $\mathcal{D}_i(\mathbf{u})$ denotes the set of defining points of $\mathbf{u}_i$.

4.2 Finding a non-dominated point

A well known theorem in multi-objective optimization states that some functions are guaranteed to lead to a non-dominated point when being optimized. Such functions are called *strongly monotone* and preserve the Pareto-dominance. More formally:

▶ **Definition 2.** *A function $g : \mathcal{Y} \mapsto \mathbb{R}$ is said to be* strongly monotone *if and only if*

$$\forall (\boldsymbol{y}, \boldsymbol{y}') \in \mathcal{Y}^2, \boldsymbol{y} \preceq \boldsymbol{y}' \implies g(\boldsymbol{y}) < g(\boldsymbol{y}')$$

▶ **Theorem 2.** *Let g be a strongly monotone function and $\mathbf{u} \in \mathbb{R}^p$. Then, if problem $\{\min g(\boldsymbol{y}) : \boldsymbol{y} \in \mathcal{Y}, \boldsymbol{y} \leqq \mathbf{u}\}$ admits $\boldsymbol{y}^*$ as an optimal solution, then $\boldsymbol{y}^* \in \mathcal{Y}_N$.*

Proof. Due to the strong monotonicity of g, any point $\bar{\boldsymbol{y}} \in Y$ dominating $\boldsymbol{y}^*$ should verify $g(\bar{\boldsymbol{y}}) < g(\boldsymbol{y}^*)$. Moreover, we have $\bar{\boldsymbol{y}} \preceq \boldsymbol{y}^* \leqq \mathbf{u}$, thus $\bar{\boldsymbol{y}}$ is feasible, contradicting the optimality of $\boldsymbol{y}^*$. ◀

5 Algorithm statement

The proposed algorithm iteratively explores the search region, trying to improve the current best known value ϕ of function Φ while limiting the number of search zones to be explored, and stops when the search region becomes empty.

5.1 Exploration of the search region

Since the search region is defined as a list of search zones, each zone is investigated independently. The exploration of the zone bounded by $\mathbf{u}$ is performed by solving integer programs *over a projection of* $\mathbf{u}$. All these programs are guaranteed to be feasible, and an initial feasible solution can be provided in each case (*warm start*). These two properties usually lead to faster solution times.

First, a lower bound over the value of Φ is computed by solving the program:

$$(\Pi(\ell, \mathbf{u})) = \{\min \ \Phi(\boldsymbol{x}) : \boldsymbol{x} \in \mathcal{X}, \boldsymbol{f}_{-\ell}(\boldsymbol{x}) \prec \mathbf{u}_{-\ell}\}$$

Note that, by Definition 1 and Theorem 1, if $\mathbf{u}_\ell$ is bounded then any defining point in $\mathcal{D}_\ell(\mathbf{u})$ is feasible for problem $(\Pi(\ell, \mathbf{u}))$, which allows us to optimize this problem using a warm start. Even if the resulting optimal solution $\hat{\boldsymbol{x}}$ is not guaranteed to be efficient, it provides a lower bound on Φ over the zone delimited by $\mathbf{u}$, but also over some similar search zones as stated by the following result.

▶ **Proposition 3.** *Let $\mathbf{u}' \in \mathbb{R}^p$ be a local upper bound such that $\mathbf{u}'_{-\ell} \leqq \mathbf{u}_{-\ell}$ for some $\ell \in \{1, \ldots, p\}$. If $(\Pi(\ell, \mathbf{u}))$ admits an optimal solution $\hat{\boldsymbol{x}}$, then $\Phi(\hat{\boldsymbol{x}})$ is a lower bound for any feasible point in the zone delimited by $\mathbf{u}'$.*

Proof. Since $\mathbf{u}'_{-\ell} \leqq \mathbf{u}_{-\ell}$, every $\boldsymbol{x} \in \mathcal{X}$ whose image by $\boldsymbol{f}$ is in the zone delimited by $\mathbf{u}'$ is feasible for $\Pi(\ell, \mathbf{u})$. ◀

Note that Proposition 3 applies for $\mathbf{u}$ in particular. Therefore, when the optimal value of problem $(\Pi(\ell, \mathbf{u}))$ does not improve ϕ, all search zones delimited by local upper bounds triggering Proposition 3 can be discarded. Otherwise, we proceed to the next step aiming at identifying a candidate while possibly discarding other search zones.

For this purpose, we look for a solution that minimizes $\boldsymbol{f}_\ell$ over the same projection, while improving ϕ. This is performed by solving:

$$(P(\ell, \mathbf{u})) = \{\min \ \boldsymbol{f}_\ell(\boldsymbol{x}) : \boldsymbol{x} \in \mathcal{X}, \boldsymbol{f}_{-\ell}(\boldsymbol{x}) \prec \mathbf{u}_{-\ell}, \ \Phi(\boldsymbol{x}) < \phi\}$$

Observe that, at this stage, the optimal solution returned by problem $(\Pi(\ell, \mathbf{u}))$ can be used as a warm start for problem $(P(\ell, \mathbf{u}))$. This program does not necessarily return an

efficient solution due to the additional constraint over the value of Φ. It provides however *a lower bound on objective $\boldsymbol{f}_\ell$ over all efficient solutions that improve the estimation over the projection.* The following result exploits this property in order to discard some search zones.

▶ **Proposition 4.** *Let $\mathbf{u}' \in \mathbb{R}^p$ be a local upper bound such that $\mathbf{u}'_{-\ell} \leqq \mathbf{u}_{-\ell}$ for some $\ell \in \{1, \dots, p\}$. If $(P(\ell, \mathbf{u}))$ yields a solution $\bar{\boldsymbol{x}}$ such that $\mathbf{u}'_\ell \leq \boldsymbol{f}_\ell(\bar{\boldsymbol{x}})$, then no point in the zone delimited by $\mathbf{u}'$ improves ϕ which can thus be discarded.*

Proof. Any solution $\boldsymbol{x}$ that is feasible for $P(\ell, \mathbf{u}')$ is also feasible for $P(\ell, \mathbf{u})$. Thus, we have $\mathbf{u}'_\ell \leq \boldsymbol{f}_\ell(\bar{\boldsymbol{x}}) \leq \boldsymbol{f}_\ell(\boldsymbol{x})$, meaning that $\boldsymbol{f}(\boldsymbol{x})$ cannot belong to the zone bounded by $\mathbf{u}'$. Due to the constraint $\Phi(\boldsymbol{x}) < \phi$ in program $(P(\ell, \mathbf{u}))$, the image by $\boldsymbol{f}$ of any improving solution cannot belong to this zone. ◀

As before, Proposition 4 applies for $\mathbf{u}$ in particular.

While solution $\bar{\boldsymbol{x}}$ of $(P(\ell, \mathbf{u}))$ improves ϕ, its efficiency is not guaranteed. Therefore, we look for an efficient solution dominating $\bar{\boldsymbol{x}}$, by solving the following program that optimizes a strongly monotone function (guaranteeing efficiency by Theorem 2) and discriminates among the resulting optimal solutions by optimizing Φ.

$$\text{(OptEff)} = \{\text{lexmin } \left\{ \sum_{i=1}^{p} \boldsymbol{f}_i(\boldsymbol{x}), \Phi(\boldsymbol{x}) \right\} : \boldsymbol{x} \in \mathcal{X}, \boldsymbol{f}(\boldsymbol{x}) \leqq \boldsymbol{f}(\bar{\boldsymbol{x}})\}$$

Note that $\bar{\boldsymbol{x}}$ is feasible for this problem and can thus be used as a warm start. The solution $\boldsymbol{x}^*$ of problem (OptEff) while being efficient is no longer guaranteed to improve ϕ.

It is important to observe that the search region is reduced at each iteration. Indeed, at least $\mathbf{u}$ is discarded by application of Proposition 3 or 4, or a new non-dominated point $\boldsymbol{y}^* = \boldsymbol{f}(\boldsymbol{x}^*)$ is found in the zone bounded by $\mathbf{u}$ and the region it dominates is thus removed (by splitting the zones $\boldsymbol{y}^*$ belongs to). The convergence of the resulting algorithm (see Algorithm 1) is therefore guaranteed under standard conditions ensuring that $\mathcal{X}_E$ is finite, trivially satisfied in particular for MOCO problems.

5.2 Updating the search region

Each time a non-dominated point $\boldsymbol{y}^* = \boldsymbol{f}(\boldsymbol{x}^*)$ is found, the search region must be updated by removing the part dominated by $\boldsymbol{y}^*$. The basic operation is described in [8] and is performed in two steps. First, the zones $\boldsymbol{y}^*$ belongs to must be split by replacing the corresponding local upper bounds $\mathbf{u}$ by their p children $\mathbf{u}^{\boldsymbol{i}}$, $i \in \{1, \dots, p\}$. Second, the search zones that are redundant, *i.e.* included in others, must be discarded. Moreover, by application of Propositions 3 and 4, additional zones can be ignored.

We associate to each local upper bound $\mathbf{u}$ a lower bound on Φ denoted by $l_\Phi(\mathbf{u})$. This lower bound is iteratively updated each time Proposition 3 can be applied, *i.e.* each time $\Pi(\ell, \mathbf{u}')$ is solved for a zone $\mathbf{u}'$ such that $\mathbf{u}_{-\ell} \leqq \mathbf{u}'_{-\ell}$. If the lower bound of a search zone is worse than ϕ, the zone is discarded.

It is also important to notice that Propositions 3 and 4 can be triggered at further iterations. For this reason, we store the successive results of problems $(\Pi(\ell, u))$ and $(P(\ell, u))$ in archives denoted by $\mathcal{A}_\Pi$ and $\mathcal{A}_P$, respectively. Entries in $\mathcal{A}_\Pi$ are 3-uples of the form $(\mathbf{u}, \ell, \Phi(\hat{\boldsymbol{x}}))$ and entries in $\mathcal{A}_P$ are 3-uples of the form $(\mathbf{u}, \ell, f_\ell(\bar{\boldsymbol{x}}))$. Before adding a new child $\boldsymbol{v}$, the archives are consulted. If Propositions 3 and 4 can be applied using an entry of $\mathcal{A}_\Pi$ or $\mathcal{A}_P$, the child can be discarded. The use of balanced trees to store the content of each archive allows us to perform efficient lookups since only lower bounds that are greater than ϕ (for $\mathcal{A}_\Pi$) or greater than $\boldsymbol{v}_j$ for some $j \in \{1, \dots, p\}$ (in $\mathcal{A}_P$) are relevant.

This update procedure is described in Algorithm 2.

5.3 Selecting a search zone

To increase the impact of Propositions 3 and 4, we want to prioritize *maximal projections* of local upper bounds, *i.e.* we want to select $\mathbf{u}$ and ℓ such that there is no local upper bound $\mathbf{u}'$ such that $\mathbf{u}_{-\ell} \leqq \mathbf{u}'_{-\ell}$. Moreover, we want to select ℓ such that $\mathbf{u}_\ell$ is bounded in order to exploit Theorem 1 to provide a defining point of $\mathbf{u}_\ell$ as a starting solution for $(\Pi(\ell, \mathbf{u}))$, which is always possible except at the first iteration.

For this reason, we suggest to compute the volume of the projection:

$$h(\mathbf{u}, \ell) = \prod_{\substack{i=1 \\ i \neq \ell}}^{p} \mathbf{u}_i - \boldsymbol{y}^{\boldsymbol{I}}_i$$

where $\boldsymbol{y}^{\boldsymbol{I}}$ denotes the ideal point of (MOP) , which can be obtained by optimizing independently each objective function $\boldsymbol{f}_i$ over $\mathcal{X}$.

Then, we select the projection maximizing this volume, among the current set $\mathcal{U}$ of local upper bounds:

$$(\mathbf{u}^*, \ell) = \operatorname{argmax}_{\substack{\mathbf{u} \in \mathcal{U} \\ i \in \{1,\dots,p\}}} \{h(\mathbf{u}, i)\}$$

Algorithm 1 Optimization over the efficient set.

Input : $\mathcal{X}$, $\boldsymbol{f}$, Φ
Output : ϕ, x^{opt}
`/* Initialize the estimation of Φ, the set of non-dominated points, the list of upper bounds and the archives */`
$\phi \leftarrow \infty$, $N \leftarrow \emptyset$, $\mathcal{U} \leftarrow \{(\infty, \dots, \infty)\}$, $\mathcal{A}_\Pi \leftarrow \emptyset$, $\mathcal{A}_P \leftarrow \emptyset$
while $\mathcal{U} \neq \emptyset$ **do**
  $(\mathbf{u}^*, \ell) \leftarrow \operatorname{argmax}_{\substack{\mathbf{u} \in \mathcal{U} \\ i \in \{1,\dots,p\}}} \{h(\mathbf{u}, i)\}$
  $\hat{\boldsymbol{x}} \leftarrow \operatorname{argmin} \{\Phi(\boldsymbol{x}) : \boldsymbol{x} \in \mathcal{X},\ \boldsymbol{f}(\boldsymbol{x})_{-\ell} \prec \mathbf{u}^*_{-\ell}\}$
  $\mathcal{A}_\Pi \leftarrow \mathcal{A}_\Pi \cup \{(\mathbf{u}^*, \ell, \Phi(\hat{\boldsymbol{x}}))\}$
  if $\Phi(\hat{\boldsymbol{x}}) \geq \phi$ **then**
    $\mathcal{U} \leftarrow \mathcal{U} \setminus \{\boldsymbol{v} \in \mathcal{U},\ \boldsymbol{v}_{-\ell} \leqq \mathbf{u}^*_{-\ell}, \boldsymbol{v}_\ell \geq f_\ell(\hat{\boldsymbol{x}})\}$
  else
    $\bar{\boldsymbol{x}} \leftarrow \operatorname{argmin} \{f_\ell(\boldsymbol{x}) : \boldsymbol{x} \in \mathcal{X},\ \boldsymbol{f}_{-\ell}(\boldsymbol{x}) \prec \mathbf{u}^*_{-\ell},\ \Phi(\boldsymbol{x}) < \phi\}$
    $\boldsymbol{x}^* \leftarrow \operatorname{arglexmin} \{\sum_{i=1}^{p} f_i(\boldsymbol{x}), \Phi(\boldsymbol{x}) : \boldsymbol{x} \in \mathcal{X}, \boldsymbol{f}(\boldsymbol{x}) \leqq \boldsymbol{f}(\bar{\boldsymbol{x}})\}$
    $N \leftarrow N \cup \{\boldsymbol{x}^*\}$
    if $\Phi(\boldsymbol{x}^*) < \phi$ `// Updating the estimation`
     then
      $\phi \leftarrow \Phi(\boldsymbol{x}^*),\ x^{\text{opt}} \leftarrow \boldsymbol{x}^*$
    update$(\mathcal{U}, \phi, \mathbf{u}^*_{-\ell}, \Phi(\hat{\boldsymbol{x}}), f_\ell(\bar{\boldsymbol{x}}), \boldsymbol{x}^*, \boldsymbol{f}(\boldsymbol{x}^*))$
    $\mathcal{A}_P \leftarrow \mathcal{A}_P \cup \{(\mathbf{u}^*, \ell, f_\ell(\bar{\boldsymbol{x}}))\}$

6 Computational experiments

The evaluation of our algorithm (referred to as TV in the following), is performed using instances of standard MOCO problems where Φ is a linear combination of the decision

Algorithm 2 Updating the search region.

Input : $\mathcal{U}$, the search region to be updated
ϕ, the value of the best known solution
$\mathbf{u}^*_{-\ell}$, the explored projection
$\Phi(\hat{\boldsymbol{x}})$, the result of $\Pi(\ell, \mathbf{u}^*)$
$\bar{\boldsymbol{y}}_\ell$, the result of $P(\ell, \mathbf{u}^*)$
$\boldsymbol{x}^*$ and $\boldsymbol{y}^*$, the efficient solution and its associated point
Output : $\mathcal{U}$, the updated search region

children $\leftarrow \emptyset$
/* Computing the maximal children */
foreach $\mathbf{u} \in \mathcal{U}$ **do**
  if $\boldsymbol{y}^* \prec \mathbf{u}$ **then**
    $\mathcal{U} \leftarrow \mathcal{U} \setminus \{\mathbf{u}\}$
    foreach $i \in \{1, \dots, p\}$ **do**
      $\mathbf{u}^{\boldsymbol{i}} \leftarrow (\mathbf{u}_1, \dots, \mathbf{u}_{i-1}, \boldsymbol{y}^*_i, \mathbf{u}_{i+1}, \dots, \mathbf{u}_p)$
      /* Computing the defining points of each bounded component */
      $\mathcal{D}_j(\mathbf{u}^{\boldsymbol{i}}) \leftarrow \{\boldsymbol{y} \in \mathcal{D}_j(\mathbf{u}),\ \boldsymbol{y}_i < \boldsymbol{y}^*_i\},\ \forall j \in \{1, \dots, p\}, \mathbf{u}^{\boldsymbol{i}}_j \neq \infty$
      $\mathcal{D}_i(\mathbf{u}^{\boldsymbol{i}}) \leftarrow \{\boldsymbol{f}(\boldsymbol{x}^*)\}$
      if $\mathcal{D}_j(\mathbf{u}^{\boldsymbol{i}}) \neq \emptyset,\ \forall j \in \{1, \dots, p\},\ \mathbf{u}^{\boldsymbol{i}}_j \neq \infty$ // The child is maximal
      and $\nexists (\boldsymbol{v}, j, opt) \in \mathcal{A}_\Pi : \mathbf{u}^{\boldsymbol{i}}_{-j} \preceq \boldsymbol{v}_{-j}, opt \geq \phi$ // No archived problem triggers Proposition 3
      and $\nexists (\boldsymbol{v}, j, opt) \in \mathcal{A}_P : \mathbf{u}^{\boldsymbol{i}}_{-j} \preceq \boldsymbol{v}_{-j}, opt \geq \mathbf{u}^{\boldsymbol{i}}_j$ // No archived problem triggers Proposition 4
      then
        children $\leftarrow$ children $\cup \{\mathbf{u}^{\boldsymbol{i}}\}$
  else if $\boldsymbol{y}^* \leqq \mathbf{u}$ **then**
    /* $\boldsymbol{y}^*$ may be a new defining point for $\mathbf{u}$ */
    $\mathcal{D}_j(\mathbf{u}) \leftarrow \mathcal{D}_j(\mathbf{u}) \cup \{\boldsymbol{y}^*\}\ \forall j \in \{1, \dots, p\},\ \boldsymbol{y}^*_{-j} \prec \mathbf{u}_{-j}$
$\mathcal{U} \leftarrow \mathcal{U} \cup$ children
/* Application of the reduction rules */
foreach $\mathbf{u} \in \mathcal{U}$ **do**
  if $\mathbf{u}_{-\ell} \leqq \mathbf{u}^*_{-\ell}$ **then**
    if $\bar{\boldsymbol{y}}_\ell \geq \mathbf{u}_\ell$ **then**
      /* Proposition 4 */
      $\mathcal{U} \leftarrow \mathcal{U} \setminus \{\mathbf{u}\}$
    else
      /* Proposition 3 */
      $l_\Phi(\mathbf{u}) \leftarrow \max\{l_\Phi(\mathbf{u}), \Phi(\hat{\boldsymbol{x}})\}$
  if $l_\Phi(\mathbf{u}) \geq \phi$ **then**
    $\mathcal{U} \leftarrow \mathcal{U} \setminus \{\mathbf{u}\}$ // The lower bound is worse than the current estimation

variables, as done for most other algorithms. Moreover we propose to compare TV with a state of the art algorithm. Natural candidates are the most recent algorithms which were themselves compared to previous ones and shown to achieve the best performances. The two most recent algorithms are [1, 11]. Comparisons in [1] report significantly better results with respect to the algorithm proposed in [5]. Comparisons in [11], for which no implementation is available, report contrasted results in particular between algorithms presented in these two papers. Moreover, algorithms in [11] deal with a special case where function Φ is a weighted combination of the objectives, with at least one negative weight. Therefore, we selected the algorithm proposed in [1] (referred to as *BCS* in the following) as a reference algorithm, using the C++ implementation provided by the authors.

Experiments have been conducted on a *Linux NixOS virtual machine* (*AMD EPYC 7702 64-Core*) running at 2000 Mhz and having 32 G of RAM. The experiments are restricted to run on a single thread, but without memory limit (less than 32G). The underlying discrete solver is *IBM Cplex* 22.10. Our code is written using the Haskell programming language and a handcrafted API for Cplex. This code is available online[2]. If an instance takes more than two hours to be solved, the tested approach is considered to have timed out.

6.1 Instances

Our approach has been validated on two sets of instances that are described in this section. In each instance, the function Φ to be minimized is randomly generated in a similar way as the objective functions.

6.1.1 MOKP

Given a set of n items, each item i having p profit values v_i^j, $j \in \{1, \dots, p\}$ and a weight w_i, the *multi-objective knapsack problem* (MOKP) consists of selecting a subset of items considering the total values on each objective, without exceeding a certain weight capacity W. This problem can be stated as:

$$(MOKP) \quad \begin{cases} \max & f_j(x) = \sum_{i=1}^{n} v_i^j x_i & \forall j \in \{1, \dots, p\} \\ \text{s.t.} & \sum_{i=1}^{n} w_i x_i \leq W & \\ & x_i \in \{0, 1\} & i \in \{1, \dots, n\} \end{cases}$$

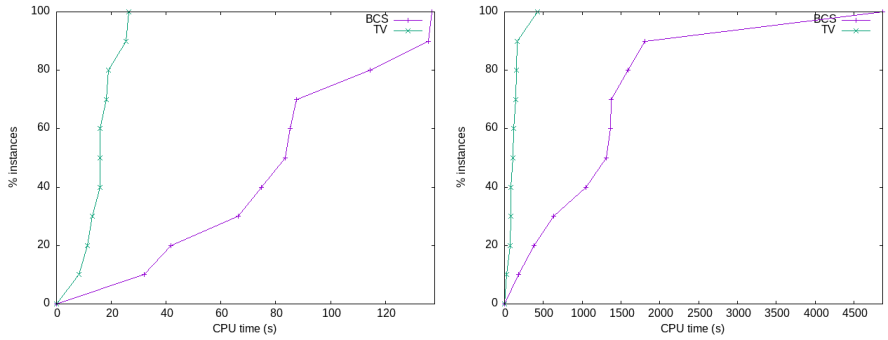
Coefficients v_i^j and w_i are uniformly sampled in $\{1, \dots, 100\}$, and W is set to $\frac{\sum_{i=1}^{n} w_i}{2}$. 10 instances of size $n = 100$ and $p = 3, 4, 5$ have been generated.

6.1.2 MOAP

Given n tasks to be performed on n machines and p costs c_{ik}^j of assigning task i to machine k, the *multi-objective assignment problem* consists of determining an assignment considering the total cost on each objective. This problem can be stated as:

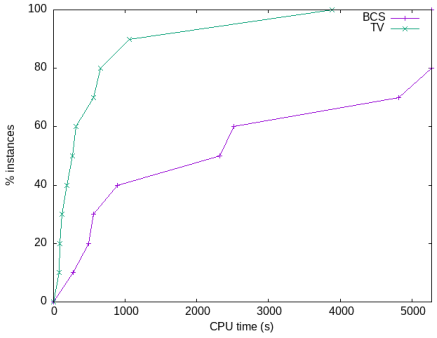
$$(MOAP) \quad \begin{cases} \min & \sum_{i=1}^{n} \sum_{k=1}^{n} c_{ik}^j x_{ik} & \forall j \in \{1, \dots, p\} \\ \text{s.t.:} & \sum_{i=1}^{n} x_{ik} = 1 & k \in \{1, \dots, n\} \\ & \sum_{k=1}^{n} x_{ik} = 1 & i \in \{1, \dots, n\} \\ & x_{ik} \in \{0, 1\} & i \in \{1, \dots, n\}, k \in \{1, \dots,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

[2] `https://github.com/tambysatya/EfficientSetOptmizer`



(a) $p = 3$.

(b) $p = 4$.



(c) $p = 5$.

Figure 1 Performance profiles on multi-objective knapsack problem (higher is better).

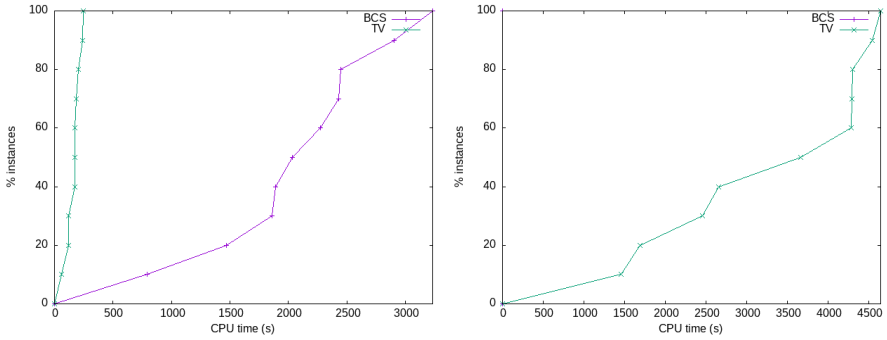
Coefficients c_{ik}^j are uniformly sampled in $\{1, \ldots, 25\}$. 10 instances of size $n = 30$ and $p = 3, 4$ have been generated.

6.2 Analysis

We first propose a comparative analysis of the CPU time required by *BCS* and *TV* on both families of instances. Performance profiles are reported in Figures 1 and 2. These plots represent the percentages of instances solved in less than t seconds, for $t \leq 7200$s. We can see that *TV* clearly outperforms *BCS* on both *MOKP* and *MOAP* instances. In particular, on the tri-objective *MOKP* and on all *MOAP* instances, *TV* solves each instance faster than the most fastly solved instance by *BCS*. In addition, *BCS* is unable to solve any instance of *MOAP* with 4 objectives and two instances of *MOKP* with 5 objectives while *TV* solves all of them in less than two hours. For the *MOKP* with 4 objectives, *TV* solves all instances in less than 500 seconds while *BCS* solves only 2 of these in this timelapse.

Second, *BCS* and *TV* are evaluated according to several measures in Tables 1 and 2. For both algorithms, the average *cpu-time*, *number of iterations* and the *number of generated non-dominated points* are presented. Additional information is reported for *TV*: the *maximum and average size of the search region*, the *percentage of zones that are discarded by reduction rules induced by Propositions 3 and 4 and using the archives*.

The average CPU time spent on each test set obviously matches the observations made from the performance profiles, validating the performance of *TV* against *BCS*, and will thus not be discussed. The number of iterations required to compute the optimum for each test set also shows that our approach converge faster. These two measures are inter-related, especially since each iteration of *BCS* involves solving problems with disjunctive constraints which are likely more difficult to be optimized and which can be infeasible while *TV* solves



(a) $p = 3$. **(b)** $p = 4$.

Figure 2 Performance profiles on multi-objective assignment problem (higher is better).

Table 1 Performance measures for *MOKP*

						mean			
p	n		CPU (s)	#It	$\|N\|$	$\|\mathcal{U}\|$max	$\|\mathcal{U}\|$avg	Reductions (%)	Archive (%)
3	100	TV	16.9	218.3	172.7	91.8	44.67	27.65	0.50
		BCS	85.8	287.0	155.5				
4	100	TV	135.8	1003.8	560.5	1237.4	696.02	14.63	0.88
		BCS	1457.1	1242.7	436.3				
5	100	TV	720.3	3123.2	1071.9	12493.2	6852.99	8.38	0.86
		BCS	-						

Table 2 Performance measures for *MOAP*.

						mean			
p	$n \times n$		CPU (s)	#It	$\|N\|$	$\|\mathcal{U}\|$max	$\|\mathcal{U}\|$avg	Reductions (%)	Archive (%)
3	30 × 30	TV	169.8	561.9	475.5	177.3	93.78	25.41	0.32
		BCS	2134.0	2042.7	1020.5				
4	30 × 30	TV	3396.0	5140.9	3267.5	10609.4	5319.58	13.88	0.33
		BCS	-						

only feasible problems that are augmented with budget constraints only. Regarding MOKP, we can observe that, while converging faster than *BCS*, *TV* generates a slightly larger number non-dominated points (for $p = 3, 4$). This is no longer true for MOAP, where *BCS* requires the generation of about four times more non-dominated points (for $p = 3$).

To perform a more detailed analysis of *TV*, several points must be discussed. First, the maximum and average size of the current search region remains "reasonable", suggesting that time is mainly spent in exploring zones and justifying our aim of reducing the number of calls to the underlying discrete solver and helping it by providing feasible problems for which initial feasible solutions are also provided (warm start). Second, we can see that reduction rules are quite efficient in particular for tri-criteria case where about a quarter of the children zones are discarded, both for MOKP and MOAP. As expected given the conditions for triggering these rules, this proportion decreases for 4 objectives (around 15%) to become less than 10% when $p = 5$. Conversely, despite being significantly smaller, the proportion of zones that are discarded using the archives is rather stable when the number of objectives grows.

7 Conclusion

While strongly relying on mechanisms developed in [14] that already proved quite successful for problem (MOP) (reduction rules to reject zones without probing them, or providing an initial solution for every integer program to be solved), this algorithm, proposed for problem (MOP_E), takes advantage of new results, notably the computation of local lower bounds on the function to be minimized over the efficient set. Additional reduction rules are thus proposed, allowing pruning the search space and converging faster. Our experiments on the multi-objective knapsack and assignment problems show promising results since this approach seems to perform significantly better than the state of the art algorithms.

Besides technically improving the current method, further works may concern studying the optimization of specific functions. For instance, a natural extension would be to apply this method to the computation of the nadir point, whose components are the worst possible values taken by the non-dominated points. Our perspective in this respect is to make use of specific properties of this point so as to adapt our approach to this problem.

References

1 Natashia Boland, Hadi Charkhgard, and Martin Savelsbergh. A new method for optimizing a linear function over the efficient set of a multiobjective integer program. *European Journal of Operational Research*, 260(3):904–919, 2017.

2 Natashia Boland, Hadi Charkhgard, and Martin W. P. Savelsbergh. The *L*-shape search method for triobjective integer programming. *Mathematical Programming Computation*, 8(2):217–251, 2016.

3 Djamal Chaabane and Marc Pirlot. A method for optimizing over the integer efficient set. *Journal of Industrial and Management Optimization*, 6(4):811–823, 2010.

4 Kerstin Dächert, Kathrin Klamroth, Renaud Lacour, and Daniel Vanderpooten. Efficient computation of the search region in multi-objective optimization. *European Journal of Operational Research*, 260(3):841–855, 2017.

5 Jesus Jorge. An algorithm for optimizing a linear function over an integer efficient set. *European Journal of Operational Research*, 195(3):98–103, 2009.

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

7 Gokhan Kirlik and Serpil Sayin. Computing the nadir point for multiobjective discrete optimization problems. *Journal of Global Optimization*, 62(1):79–99, 2015.

8 Kathrin Klamroth, Renaud Lacour, and Daniel Vanderpooten. On the representation of the search region in multi-objective optimization. *European Journal of Operational Research*, 245(3):767–778, 2015.

9 Dieter Klein and Edward L. Hannan. An algorithm for the multiple objective integer linear programming problem. *European Journal of Operational Research*, 9(4):378–385, 1982.

10 Murat Köksalan and Banu Lokman. Finding nadir points in multi-objective integer programs. *Journal of Global Optimization*, 62(1):55–77, 2015.

11 Banu Lokman. Optimizing a linear function over the nondominated set of multiobjective integer programs. *International Transactions in Operational Research*, 28(4):2248–2267, 2021.

12 Banu Lokman and Murat Köksalan. Finding all nondominated points of multi-objective integer programs. *Journal of Global Optimization*, 57(2):347–365, 2013.

13 John Sylva and Alejandro Crema. A method for finding the set of non-dominated vectors for multiple objective integer linear programs. *European Journal of Operational Research*, 158(1):46–55, 2004.

14 Satya Tamby and Daniel Vanderpooten. Enumeration of the nondominated set of multiobjective discrete optimization problems. *INFORMS Journal on Computing*, 33(1):72–85, 2021.