RAIRO-Oper. Res. 55 (2021) S2471–S2485
https://doi.org/10.1051/ro/2020139

RAIRO Operations Research
www.rairo-ro.org

AN ITERATIVE VERTEX ENUMERATION METHOD FOR OBJECTIVE SPACE BASED VECTOR OPTIMIZATION ALGORITHMS

İrfan Caner Kaya and Firdevs Ulus*

Abstract. An application area of vertex enumeration problem (VEP) is the usage within objective space based linear/convex vector optimization algorithms whose aim is to generate (an approximation of) the Pareto frontier. In such algorithms, VEP, which is defined in the objective space, is solved in each iteration and it has a special structure. Namely, the recession cone of the polyhedron to be generated is the ordering cone. We consider and give a detailed description of a vertex enumeration procedure, which iterates by calling a modified "double description (DD) method" that works for such unbounded polyhedrons. We employ this procedure as a function of an existing objective space based vector optimization algorithm (Algorithm 1); and test the performance of it for randomly generated linear multiobjective optimization problems. We compare the efficiency of this procedure with another existing DD method as well as with the current vertex enumeration subroutine of Algorithm 1. We observe that the modified procedure excels the others especially as the dimension of the vertex enumeration problem (the number of objectives of the corresponding multiobjective problem) increases.

Mathematics Subject Classification. 90C29, 52B11, 68W27.

Received June 30, 2020. Accepted December 1, 2020.

1. Introduction

A polyhedron $P \subseteq \mathbb{R}^d$ can be represented as intersection of finitely many halfspaces or as convex hull of its vertices added to the conic hull of its extreme directions. The problem of computing the vertex representation of P from its halfspace representation is called the *vertex enumeration problem* (VEP). VEP has been studied for many years, starting at latest from the 1950s, see for instance [5, 25]. The difficulty of the problem is known [4, 19] and there are many studies to propose efficient algorithms or to improve the efficiency of existing ones [2, 13, 17, 28].

The vertex enumeration problem as defined above is sometimes called the *off-line* VEP, whereas finding the vertices of a polyhedron P'' given as intersection of a single halfspace H and another polyhedron P' whose vertex representation is known is called the *on-line* VEP. Note that an on-line vertex enumeration algorithm can be called repetitively in order to solve an off-line VEP. Different from those, there are also (simplex-type) pivoting algorithms that solve off-line VEP directly, see for instance [1, 3, 5].

The problem of finding vertices of a polyhedron is fundamental and it is a base of some other algorithms including the simplex algorithm to solve linear programs and outer approximation algorithms for DC

Keywords. Vertex enumeration, multiobjective optimization, vector optimization, polyhedral approximation.

Bilkent University, Department of Industrial Engineering, Bilkent, Ankara 06800, Turkey.
*Corresponding author: firdevs@bilkent.edu.tr

programming problems, see for instance [29]. It has also been an important part of objective space based algorithms designed to solve linear [7, 12, 18] or convex [15, 24] multiobjective and vector optimization problems (VOPs). In general, these algorithms aim to find (a polyhedral approximation of) the set of all nondominated points in the objective space, namely the Pareto frontier. Clearly, for a VOP with d objectives, the objective space is $\mathbb{R}^d$. In each iteration of such algorithm, an on-line vertex enumeration problem of size d has to be solved. Note that for a VOP with an ordering cone K, the nondominated (K-minimal) points of the image of the feasible region is the same as that of the *upper (extended) image*, which is K added to the image of the feasible region. Indeed, for these algorithms, the idea is to approximate the upper image by inner and outer polyhedral sets.

Solving sequential online VEPs is also used in order to solve data envelopment analysis (DEA) problems recently. Recall that DEA is used to compute the efficiency of decision-making units (DMUs) with common inputs and outputs (for a review, see [10, 11]), and it has many application areas including, banking, healthcare, energy and environmental sciences, agriculture, see for instance the survey papers [20, 30]. In [16], Ehrgott, Hasannasab and Raith proposed an algorithm in order to generate the extreme points and facets of the efficient frontier of data envelopment analysis (DEA) problems using the geometric duality theory for linear multiobjective optimization problems (MOPs). Accordingly, instead of relying on solving a linear program for each DMU as many other DEA solution approaches from the literature do, their algorithm is based on solving online VEPs in each iteration and does not solve any LP. It is demonstrated in [16] that their algorithm is computationally comparable with the standard DEA approach for some real life problems and it is faster than that for large-scale artificial data sets for which the percentage of efficient DMUs is rather small.

Non-pivoting on-line vertex enumeration algorithms in the literature are generally designed to find the vertices of bounded polyhedrons, see for instance [9]. However, for aforementioned vector optimization algorithms, one needs to find the vertices of unbounded polyhedrons since the upper image of a VOP is an unbounded polyhedron whose recession cone is (at least) the ordering cone. In 2010, using projective geometry, Burton and Ozlen [8] proposed a vertex enumeration method which works for unbounded polyhedrons with known recession cones.

In this study, we first consider the "standard" double description (DD) method proposed for bounded polyhedrons. The detailed description is provided in [17], where the problem is given in an equivalent form of enumerating the extreme directions of a polyhedral cone. Note that the DD method is designed to solve the on-line vertex enumeration problem and by employing it in each iteration, one can solve an off-line VEP as well. In particular, one needs to start with a sufficiently large bounded polyhedron P^0 that contains the set of all vertices of P. This method can be applied both for bounded or unbounded P for which the recession cone is not necessarily known. As long as the vertices of P are known to be included in P^0, one can use DD method iteratively and at the end of the final iteration, one needs to get rid of the vertices that are on the boundary of P^0. The main difficulty in this approach is to find such P^0. Also, taking P^0 too large may result in numerical issues when implemented.

We then consider the modified DD method (proposed in [8]), which works for unbounded P whose recession cone, say K, is known. We provide a detailed description of the algorithm in the Euclidean space (instead of the oriented projective space). Note that the main difference of the modified method is the use of extreme directions of the current approximation P' in the computations of the vertices of the updated polyhedron P''. Clearly, the modified DD method can also be called iteratively in order to solve an off-line VEP. This time one needs to start with a single vertex $v_0 \in \mathbb{R}^d$ such that the initial polyhedron $P^0 := v^0 + K$ contains the polyhedron to be computed. For computing the upper image of a multiobjective optimization problem (MOP), where the ordering cone is the nonegative cone, *ideal point* is the natural candidate for this initial vertex v^0.

We implement the standard and the modified DD methods, and compared their efficiencies through computational tests. Indeed, we employ both methods as a subroutine within a MATLAB implementation of the objective space based convex vector optimization algorithm proposed in [24]. The current implementation of the algorithm calls in each iteration a vertex enumeration procedure (*vert*) which mainly uses the "qhull" function of MATLAB [6]. Note that this vertex enumeration procedure is first employed within the MATLAB implementation of *bensolve*, which is a linear vector optimization solver, see [22, 23]. We test the performances of the original and the modified DD methods together with *vert* through randomly generated linear MOPs.

In Section 2, we provide preliminaries on basic convex analysis and on convex VOPs as well as the convex VOP algorithm proposed in [24]. The DD method and the modified variant are described in Section 3. The variants of the convex VOP algorithm using different DD methods are explained in Section 4. The computational results are presented and discussed in Section 5. We conclude our discussion in Section 6.

2. Preliminaries

The boundary, the interior, the convex hull and the conic hull of a subset $S \subseteq \mathbb{R}^d$ is denoted by $\operatorname{bd} S$, $\operatorname{int} S$, $\operatorname{conv} S$, $\operatorname{cone} S$, respectively.

Let S be a convex subset of $\mathbb{R}^d$ and $F \subseteq S$ be a convex subset. If $\lambda y^1 + (1-\lambda)y^2 \in F$ for some $0 < \lambda < 1$ holds only if both y^1 and y^2 are elements of F, then F is a *face* of S. A zero dimensional face is an *extreme point*, a one dimensional face is an *edge* and a $d-1$ dimensional face is a *facet* of S, see [27].

For a subset S of $\mathbb{R}^d$, $z \in \mathbb{R}^d \setminus \{0\}$ is a *recession direction* (or simply *direction*) of S, if $y + \gamma z \in S$ for all $\gamma \geq 0$ and $y \in S$. The set of all recession directions constitute the *recession cone* of S which is denoted by $\operatorname{recc} S$. A recession direction $z \in \operatorname{recc} S \setminus \{0\}$ of convex set S is said to be an *extreme direction* of S, if $\{v + rz \in \mathbb{R}^d \mid r \geq 0\}$ is a face for some extreme point v of S. $S \subseteq \mathbb{R}^d$ is *bounded* if $\operatorname{recc} S = \{0\}$.

A closed convex pointed solid cone $K \subseteq \mathbb{R}^d$ defines a partial order on $\mathbb{R}^d$ as follows: $a \leq_K b \iff b - a \in K$. Throughout, $\mathbb{R}^d_+ := \{y \in \mathbb{R}^d \mid y_i \geq 0, i = 1, \dots, d\}$ is the nonnegative cone in $\mathbb{R}^d$, e^i is the unit vector in $\mathbb{R}^d$ with ith component being 1.

2.1. Representations of a convex polyhedron

If a convex set P can be written as $P = \{y \in \mathbb{R}^d \mid A^T y \geq b\}$, where $A \in \mathbb{R}^{d \times k}$ and $b \in \mathbb{R}^k$, then it is called a *polyhedral convex set* or a *convex polyhedron*. Note that P is intersection of finitely many half-spaces, namely,

$$P = \bigcap_{i=1}^{k} \{y \in \mathbb{R}^d \mid a_i^T y \geq b_i\}, \tag{2.1}$$

where $a_i \in \mathbb{R}^d$ is the ith column of matrix A and $b_i \in \mathbb{R}$ is the ith component of b. The representation given in (2.1) (with the assumption that there are no redundant inequalities) is called *H-representation* or *halfspace representation* of P. On the other hand, if P has at least one extreme point, it can also be represented as the convex hull of all its extreme points added to the conic hull of all its extreme directions. To be more precise, let V be the finite set of all extreme points (vertices) of P and D be the finite set of all extreme directions of P. Then, P can be written as

$$P = \operatorname{conv} V + \operatorname{cone} \operatorname{conv} D. \tag{2.2}$$

The representation given by (2.2) of P is called the *V-representation* or the *vertex representation* of the polyhedral convex set P. The problem of finding the *V-representation* of a set given its *H-representation* is called the *vertex enumeration problem*.

2.2. Convex vector optimization and an approximation algorithm

A convex vector optimization problem is given by

$$\text{minimize } F(x), \text{ with respect to } \leq_K, \text{ subject to } x \in \mathcal{X}, \tag{P}$$

where $K \in \mathbb{R}^d$ is solid, pointed, polyhedral convex ordering cone, $F : \mathbb{R}^n \to \mathbb{R}^d$ is a K-convex continuous function and the feasible set $\mathcal{X} \subseteq \mathbb{R}^n$ is a convex set. The image of the feasible set is given by $F(\mathcal{X}) = \{F(x) \in \mathbb{R}^d \mid x \in \mathcal{X}\}$ and the set $\mathcal{P} := \operatorname{cl}(F(\mathcal{X}) + K)$ is called the *upper (extended) image* of (P). It is known that $\mathcal{P}$ is convex and closed. (P) is said to be a bounded problem if there exists $y \in \mathbb{R}^d$ such that $\mathcal{P} \subseteq \{y\} + K$.

If the ordering cone is the nonnegative cone, then (P) is a multiobjective optimization problem and the *ideal point* of problem (P) can be found by minimizing each objective F_i, for $i = 1, \dots, d$ over feasible set $\mathcal{X}$ as long as the corresponding single objective optimization problems have finite optimal objective values. More specifically, let $y_i := \min\{F_i(x) \mid x \in \mathcal{X}\}$. Then, $y^I := (y_1, \dots, y_d)^T$ is the ideal point of (P). The problem is bounded if $y^I \in \mathbb{R}^d$.

For VOPs, there are different solution concepts as there is not necessarily a unique "solution" that minimizes all the objective functions simultaneously. Some of the solution concepts are as follows: A point $y \in F(\mathcal{X})$ in the image set is said to be a *K-minimal (non-dominated)* point if

$$(\{y\} - K) \cap F(\mathcal{X}) = \{y\}.$$

Similarly $y \in F(\mathcal{X})$ is said to be a *weakly K-minimal (weakly non-dominated)* point if

$$(\{y\} - \operatorname{int} K) \cap F(\mathcal{X}) = \emptyset.$$

A feasible point $x \in \mathcal{X}$ is said to be a *(weakly) efficient solution* if $F(x)$ is a (weakly) non-dominated point of $F(\mathcal{X})$.

In some applications of VOPs, it is important to generate the set of all (weakly) non-dominated points of $F(\mathcal{X})$, which is a subset of the boundary of the upper image. When the problem is linear, then it is possible to generate (the set of all extreme points of) the upper image, see for instance [7, 18]. If the problem is nonlinear convex, it is not possible to generate the set of all (weakly) non-dominated points in general. Instead, there are objective space based algorithms that can generate polyhedral approximations to the upper image as in [15, 24].

The general idea of such an algorithm is as follows. It starts with finding an initial outer approximation P^0 of $\mathcal{P}$. In case of a MOP, this can be done by finding the ideal point y^I of problem (P). Then, the initial outer approximation of $\mathcal{P}$ is $P^0 := \{y^I\} + \mathbb{R}^d_+$.

At ith iteration of the algorithm, the first step is to find the vertices of the current outer approximation P^{i-1}. Next, for a vertex v of P^{i-1}, single objective convex optimization problem, namely the Pascoletti-Serafini scalarization [26], given by

$$\text{minimize} \quad \alpha \quad \text{subject to} \quad F(x) \leq_K v + \alpha k, \;\; x \in \mathcal{X}, \qquad (\mathrm{P}(v))$$

for some fixed $k \in \operatorname{int} K$, is solved. Note that this problem finds point $y^v := v + \alpha^v k$ on $\operatorname{bd} \mathcal{P}$ that is "closest" (through the fixed direction k) to v, where α^v is the optimal objective function value of the program. Moreover, optimal solution x^v is known to be weakly efficient.

If $\alpha^v > \epsilon$, where ϵ is the predetermined approximation error, then by using the dual solution of this scalar convex optimization problem, one finds a supporting hyperplane of $\mathcal{P}$ at y^v. More specifically, if $w \in \mathbb{R}^d$ denotes the dual variable corresponding to the first set of constraints $F(x) \leq_K v + \alpha k$; and w^v is the dual optimal solution, then

$$h^v := \{y \in \mathbb{R}^d |\; (w^v)^T y = (w^v)^T y^v\}$$

supports $\mathcal{P}$ at y^v and

$$H^v := \{y \in \mathbb{R}^d |\; (w^v)^T y \geq (w^v)^T y^v\}$$

is the corresponding halfspace that contains $\mathcal{P}$ [24]. After computing H^v, the outer approximation of the upper image is updated as $P^i := P^{i-1} \cap H^v$ and the ith iteration is completed.

If $\alpha^v \leq \epsilon$, then the algorithm continues checking other vertices of the current outer approximation. The algorithm stops when all the vertices are in ϵ-distance to the upper image. One can see the books [14, 21] for details of the multiobjective/vector optimization theory and of objective space based (also referred to as Benson-type [7]) algorithms. We provide the pseudo-code of the algorithm as proposed in [24], see Algorithm 1.

There are different variants of the algorithm in the literature. For example, in each iteration of the variant proposed in [15], the direction parameter (k in (P(v))) is chosen in a different way depending on v, and the supporting hyperplane is constructed using the differentials of the objective functions instead of using the dual optimal solution of (P(v)).

Algorithm 1. An objective space based convex VOP algorithm.

```
1: Compute an initial outer approximation $P^0$ of $\mathcal{P}$, initialize $\bar{\mathcal{X}}$ accordingly;
2: $i = 0$;
3: repeat
4:    continue = 0;
5:    Compute the vertices $V^i$ of $P^i$;
6:    for all $v \in V^i$ do
7:       Solve (P(v)), let optimal solution be $(x^v, \alpha^v)$ and dual optimal solution be $w^v$;
8:       $\bar{\mathcal{X}} \leftarrow \bar{\mathcal{X}} \cup \{x^v\}$
9:       if $\alpha^v > \epsilon$ then
10:         $i \leftarrow i + 1$;
11:         $P^i = P^{i-1} \cap H^v$;
12:         continue = 1;
13:         break;
14:      end if
15:   end for
16: until continue = 0
17: return  $\bar{\mathcal{X}}$: Set of weakly efficient solutions
            $P^i$: Polyhedral outer approximation to the upper image.
```

For a MOP, the initialization can be performed as follows:

```
1: Compute $y_i := \min\{f_i(x) \mid x \in \mathcal{X}\}$, $x^i := \arg\min\{f_i(x) \mid x \in \mathcal{X}\}$;
2: $y^I := (y_1, \dots, y_d)^T$, $P^0 = \{y^I\} + \mathbb{R}^d_+$, $\bar{\mathcal{X}} = \{x^1, \dots, x^d\}$;
```

Remark 2.1. Note that at each iteration of the algorithm, the first step is to solve an on-line VEP and these VEPs are in a special form. The polyhedron to be found is unbounded since $\mathcal{P}$ is an unbounded set. Moreover, assuming that the VOP is bounded, the recession cone of the polyhedron is the ordering cone (and not larger than that). Hence the extreme directions of the recession cone are the extreme directions of K, namely $Z := \{z^1, \dots, z^m\}$. In case of a MOP, they are nothing but the unit directions, $\{e^1, \dots, e^d\}$.

3. The Double Description (DD) method

We first describe the DD method which works for bounded polyhedrons and then describe a modification of it which works for unbounded polyhedrons with known recession cones.

For both methods, let P' be a convex polyhedron, H be a halfspace given by $H := \{x \in \mathbb{R}^d \mid a^T x \geq b\}$ for some $a \in \mathbb{R}^d \setminus \{0\}$ and $b \in \mathbb{R}$, and $h := \operatorname{bd} H$ be the hyperplane given by the boundary of H.

3.1. The DD method

In this section, we describe the DD method, as provided in [8, 12].

Let P' be bounded, V' be the set of its vertices and F' be the set of its facets. The following is a useful definition in order to describe the method.

Definition 3.1. If vertex $v \in V'$ is on facet $f \in F'$; then v is said to be an *adjacent vertex* of facet f and f is said to be an *adjacent facet* of vertex v.

For a vertex v, let F_v denote the set of all adjacent facets of v, and for a facet f, let V_f denote the set of all adjacent vertices of f. These sets are called the adjacency lists. It is assumed that the sets V', F' as well as V_f, F_v for all $v \in V'$ and $f \in F'$ are known.

As it is an important subroutine in DD method, we first describe a procedure to check if there is an edge between given two vertices of polyhedron P'. Let $v^+, v^- \in V'$ be two vertices. In order to check if there is an edge between them, one considers the set of facets which are both adjacent to v^+ and v^-. Then, for each facet in this set, one considers the adjacent vertices of it. If the intersection of the set of vertices over all these facets consists of only v^+ and v^- then, there is an edge between the two; otherwise, there is no edge between them, see Lemma 5.3 of [8] and [17]. Procedure 1 is the pseudo-code for the *isedge* function, which takes polyhedron P' and vertices v^+, v^- as its input; and returns the set of facets that contains the edge between them if there is any or returns empty set otherwise. Note that with P' being an input we mean that there is an access to V', F' as well as V_f and F_v for all $f \in F'$ and $v \in V'$. This will be the case for all the procedures described later as well.

Procedure 1. *isedge*(P', v^+, v^-).

```
1: Let $F^{\pm} := F_{v^+} \cap F_{v^-}$;
2: Let $V^{\pm} := V'$;
3: for $i = 1 : |F^{\pm}|$ do
4:     Let $f^i$ be the $i$th facet in $F^{\pm}$;
5:     $V^{\pm} \leftarrow V^{\pm} \cap V_{f^i}$;
6: end for
7: if $V^{\pm} = \{v^+, v^-\}$ then
8:     return $F^{\pm}$
9: else
10:    return $\emptyset$
11: end if
```

The idea of the double description method is as follows: First, it checks if each vertex v in V' is in the interior of H, on the boundary h of H, or not included in H. Clearly, the ones that are not in H will not be a vertex of the updated polyhedron anymore. As long as there exists at least one vertex that is included in H and there exists at least one vertex that is not included in it then, the algorithm considers each couple of vertices v^+ and v^- in V', where $v^+ \in \operatorname{int} H$ and $v^- \notin H$, and checks if there is an edge between v^+ and v^-. For the couples that form an edge, a new vertex is found by intersecting the edge with hyperplane h. This new vertex is a vertex of the updated polyhedron P'' and h is a facet of P''. Each time a new vertex is found, the adjacency lists are updated accordingly.

The pseudo-code for the double description method is given by Procedure 2. The function *onlinevert* takes polyhedron P' and halfspace H as its input and returns the updated polyhedron $P'' = P' \cap H$.

3.2. The modified DD method

We consider a modified double description method which works for unbounded polyhedrons with known recession cones. This is proposed and described in [8], using oriented projective geometry. Here, we provide a detailed description in Euclidean space. Let V' be the set of vertices, F' be the set of facets and $Z = \{z^1, \ldots, z^m\}$ be the set of extreme directions of P'. We assume that the recession cone of the updated polyhedron $P'' = P' \cap H$ is also $\operatorname{cone\,conv} Z =: K$. Note that this is the case if this method is used to compute the vertices of (an approximation of) the upper image of a VOP, see Remark 2.1.

In order to describe the modified DD method, in addition to Definition 3.1, we need the following:

Definition 3.2. Let P be an unbounded polyhedron. An extreme direction $z \neq 0$ of $\operatorname{recc} P$ is said to be an *adjacent direction* of facet f of P if there exists an extreme point v of P such that $\{v + \gamma z |\ \gamma \geq 0\}$ forms an edge of P and is on facet f. Symmetrically, f is said to be an *adjacent facet* of direction z.

Remark 3.3. As before, F_v denotes the set of all adjacent facets of vertex v. Different from the previous case, for this method, V_f denotes the set of adjacent vertices together with the set of adjacent directions of facet f. Moreover, F_z is the set of adjacent facets of extreme direction z. Here, the extreme directions are treated as vertices. Indeed, they are vertices of the polyhedron when it is described using the oriented projective geometry,

Procedure 2. *onlinevert*(P', H).

```
Initialize $V^0, V^+, V^- := \emptyset$;
for all $v \in V'$ do
    if $a^T v > b$ ($i.e.$ $v \in \operatorname{int} H$) then
        $V^+ \leftarrow V^+ \cup \{v\}$;
    else if $a^T v = b$ ($i.e.$ $v \in h$) then
        $V^0 \leftarrow V^0 \cup \{v\}$;
    else if $a^T v < b$ ($i.e.$ $v \notin H$) then
        $V^- \leftarrow V^- \cup \{v\}$;
    end if
end for
if $V^+ \cup V^0 = V'$ then
    return $P'' = P'$;
else if $V^- = V'$ then
    return $P'' = \emptyset$;
else
    $F' \leftarrow F' \cup \{h\}$, $V_h = \emptyset$;
    for all $v \in V^0$ do
        $V_h \leftarrow V_h \cup \{v\}$ and $F_v \leftarrow F_v \cup \{h\}$;
    end for
    for all $v^+ \in V^+$ do
        for all $v^- \in V^-$ do
            if $F^\pm := isedge(P', v^+, v^-) \neq \emptyset$ then
                Find $v' := [v^+, v^-] \cap h$
                if $v' \notin V'$ then
                    $V' \leftarrow V' \cup \{v'\}$, $V_h \leftarrow V_h \cup \{v'\}$ and $F_{v'} = F^\pm \cup \{h\}$;
                else
                    $F_{v'} \leftarrow F_{v'} \cup F^\pm \cup \{h\}$ and $V_h \leftarrow V_h \cup \{v'\}$;
                end if
                for all $f \in F_\pm$ do
                    $V_f \leftarrow V_f \cup \{v'\}$
                end for
            end if
        end for
    end for
end if
$V'' = V' \setminus V^-$ and $F'' = \cup_{v \in V''} F_v$;
$V_f \leftarrow V_f \setminus V^-$ for $f \in F''$;
return $P''$ with vertices $V''$, facets $F''$ and respective adjacency lists.
```

see [8]. Hence, from now on whenever we mention adjacent vertices of a facet, we mean the union of adjacent vertices and adjacent directions of it.

Note that *isedge* function for given two vertices of polyhedron P' does not use the coordinates of the vertices but only the adjacency lists. Then, by definition of an adjacent facet of an extreme direction z and by the usage of notation V_f (the union of adjacent vertices and adjacent directions of facet f), $isedge(P', z, v)$ returns the set of facets on which $\{v + \gamma z | \ \gamma \geq 0\}$ lays if this is an edge of P'; and returns empty set otherwise, see Theorem 5.6 of [8] for the proof.

When treating the extreme directions as vertices, one needs to be careful whenever hyperplane h is parallel to some of these extreme directions. Note that if v^- is not in H, h is not parallel to z and $\{v^- + \gamma z | \ \gamma \geq 0\}$ is an edge of P'; then, h intersects with this edge at a singleton, namely at $v' := \{v^- + \gamma z | \ \gamma \geq 0\} \cap h$. Clearly, v' is a vertex of the updated polyhedron.

If h is parallel to a direction z, it does not intersect any edge of the form $\{v + \gamma z |\ \gamma \geq 0\}$. Instead, (as long as it cuts) it cuts polyhedron P' in parallel to direction z. Hence, h is an adjacent facet of direction z. Indeed, there must be a vertex v of P'' such that $\{v + \gamma z |\ \gamma \geq 0\}$ is an edge of P'' and lays on h. Similarly, z is an adjacent direction of facet h.

The general structure of the modified DD method (*onlinevert*2) is similar to *onlinevert*. The lines between 1–19 and 36–38 of Procedure 2 are exactly the same for *onlinevert*2 as well. The only difference is in the main loop and its pseudo-code can be seen in Procedure 3.

Procedure 3. *onlinevert*2(P', H) (substitution to lines 20–34 of Procedure 2).

```
20: for all $v^+ \in V^+ \cup Z$ do
21:   if $v^+ \in Z$ and $a^T v^+ = 0$ ($h$ is parallel to $v^+$) then
22:     $V_h \leftarrow V_h \cup \{v^+\}, F_{v^+} = F_{v^+} \cup \{h\}$;
23:   else
24:     for all $v^- \in V^-$ do
25:       if $F^{\pm} := isedge(P', v^+, v^-) \neq \emptyset$ then
26:         if $v^+ \in V^+$ then
27:           Find $v' := [v^+, v^-] \cap h$;
28:         else
29:           Find $v' := \{v^- + \gamma v^+ |\ \gamma \geq 0\} \cap h$;
30:         end if
31:         if $v' \notin V'$ then
32:           $V' \leftarrow V' \cup \{v'\}$, $V_h \leftarrow V_h \cup \{v'\}$ and $F_{v'} = F^{\pm} \cup \{h\}$;
33:         else
34:           $F_{v'} \leftarrow F_{v'} \cup F^{\pm} \cup \{h\}$ and $V_h \leftarrow V_h \cup \{v'\}$;
35:         end if
36:         for all $f \in F_{\pm}$ do
37:           $V_f \leftarrow V_f \cup \{v'\}$;
38:         end for
39:       end if
40:     end for
41:   end if
42: end for
```

The main loop goes over all vertices that are in int H and over all directions Z (line 20). If v^+ is one of the directions, say $z \in Z$, and if z is parallel to h, then z is added as an adjacent "direction" (vertex) of h and h is added as an adjacent facet of z (lines 21 and 22). Otherwise, *i.e.*, when $v^+ \in V^+$ or when v^+ is a direction which is not parallel to h; the algorithm goes over all vertices v^- that are not included in H and checks if there is an edge formed by v^+ and v^- (line 25). If there exists an edge and if v^+ is a vertex (not a direction), then the line segment $[v^+, v^-]$ intersects h at a single point v' (line 27). If v^+ is a direction and together with v^- it forms an edge, then $\{v^- + \gamma v^+ |\ \gamma \geq 0\}$ intersects h at a single point v' (line 29). In both cases, v' is a vertex of the updated polyhedron. The adjacency lists are updated in the same way as it is done for *onlinevert* (lines 31–38). As a final step after the main loop, the final set of vertices and facets together with adjacency lists are updated as it is done in *onlinevert* (see lines 36 and 37 of Procedure 2). Then, the updated polyhedron P'' is returned (line 38 of Procedure 2).

4. VOP algorithms with DD methods

In the current implementation of Algorithm 1 [24], vertex enumeration problems are solved using a MATLAB function (*vert*) written for MATLAB implementation of an objective space based (Benson-type) linear vector optimization solver *bensolve* [22]. Even though an on-line vertex enumeration problem is solved at each iteration of Benson-type algorithms, *vert* solves an off-line vertex enumeration problem. Hence, at each iteration, it computes all the vertices from an H-representation of the current outer approximation even though many vertices are already found in earlier iterations.

The double description methods described in Section 3 are used in order to solve the on-line vertex enumeration problem. For Algorithm 1, online vetex enumeration subroutine can be called repetitively in order to compute the vertices of the outer approximation in each iteration. Below, we describe two variants of Algorithm 1 that are using *onlinevert* and *onlinevert*2, respectively.

Throughout, we assume that the double description of the ordering cone K is known. In other words, adjacent facets $\tilde{F}_{z^i}$ for each extreme direction $z^i \in Z = \{z^1, \dots, z^m\}$ and adjacent directions $\tilde{V}_{\tilde{f}^i}$ for each facet, say $\tilde{f}_i \in \tilde{F} := \{\tilde{f}_1, \dots, \tilde{f}_l\}$ are known. Note that for $K = \mathbb{R}^d_+$, we have $Z = \{e^1, \dots, e^d\}$ and conic hull of any $d-1$ extreme directions from Z forms a facet of $\mathbb{R}^d_+$. More specifically, the set of all facets are $\tilde{F} := \{\tilde{f}_1, \dots, \tilde{f}_d\}$ with $\tilde{f}_i := \operatorname{cone} \operatorname{conv} \{Z \setminus \{e^i\}\}$. Then, the adjacent directions of facet $\tilde{f}_i$ is $\tilde{V}_{\tilde{f}^i} = Z \setminus \{e^i\}$ and the adjacent facets of direction e^i is $\tilde{F}_{e^i} = \tilde{F} \setminus \{\tilde{f}^i\}$. For the computational tests in Section 5, we consider MOPs where $K = \mathbb{R}^d_+$.

4.1. Variant 1: VOP algorithm with DD method

The double description method is used when the initial polyhedron is bounded. However, it can still be used in order to compute the vertices of unbounded polyhedrons. In order to do that, one needs to initialize the algorithm with a large enough initial polyhedron which guarantee to include all the vertices of the polyhedron. Note that as there are finitely many vertices, there exists a bounded polyhedron which contains all. In general any polyhedron satisfying this property can be taken as the initial one.

Recall that Algortihm 1 starts by finding an initial outer approximation P^0. For a bounded problem this can be described as $\{y\} + K$ for some $y \in \mathbb{R}^d$ and for a MOP, y can be taken as the ideal point y^I. Moreover, we assume that there exists a sufficiently large number M such that the set of all nondominated points is a subset of $P^0 := \{y\} + \operatorname{conv}\{0, Mz^1, \dots, Mz^m\}$. Indeed, if the feasible region of the problem is compact, this would be the case as the image of the feasible region in the objective space is bounded. For linear MOPs, there exists such M as long as the ideal point is finite (which may be the case even if the feasible region is not compact).

Initialization: Initial polyhedron P^0 has $m+1$ vertices, $V^0 = \{v^0, v^1, \dots, v^m\}$, where $v^0 = y$, $v^i = v^0 + Mz^i$ for $i = 1, \dots, m$. Moreover, there are $l+1$ facets, l of which correspond to the facets $\tilde{F}$ of cone K in the sense that $f^i = (\{y\} + \tilde{f}^i) \cap P^0$, for $i = 1, \dots, l$, and f^0 is given by $\operatorname{conv}\{V^0 \setminus \{v^0\}\}$. Then, $F_{v^i} = \{f^0\} \cup \{f^j |\ \tilde{f}^j \in \tilde{F}_{z^i}\}$ for $i = 1, \dots, m$ and $F_{v^0} = \{f^1, \dots, f^l\}$. Moreover, $V_{f^0} = V^0 \setminus \{v^0\}$ and $V_{f^i} = \{v^0\} \cup \{v_j |\ z_j \in \tilde{V}_{\tilde{f}^i}\}$.

Variant 1. Substitution of line 5 of Algorithm 1.

```
if $i = 0$ then
    Initialize $P^0$ as described (with vertices $V^0$ and facets $F^0$ and adjacency lists);
else
    $P^i = onlinevert(P^{i-1}, H)$;
end if
```

Variant 1. Substitution of line 17 of Algorithm 1.

```
for all $v \in V^i$ do
    if $v \in f^0$ then
        $V^i \leftarrow V^i \setminus \{v\}$;
    end if
end for
return  $\bar{\mathcal{X}}$: Set of weakly efficient solutions
        $P := \operatorname{conv} V^i + K$: Polyhedral outer approximation to the upper image.
```

The changes in the pseudo-code for this variant is given by Variant 1. Note that the vertices on facet f^0 are "artificial" by the construction of the initial polyhedron, hence the vertices on facet f^0 of P^0 are eliminated from the set of vertices of the current (last) polyhedron. This is why one needs additional lines before returning the output of the algorithm.

4.2. Variant 2: VOP algorithm with modified DD method

For this variant of Algorithm 1, we call the modified DD method in order to solve the offline VEP for unbounded polyhedrons. The structure of the main algorithm is almost the same as the previous one. The only difference is in its initialization. The changes in the pseudo-code is given in Variant 2.

Initialization: Note that $P^0 = \{y\} + K$ is the initial polyhedron. Then, the set of vertices of the initial polyhedron is $V^0 = \{y\}$. Moreover, there are l facets given by $f^i = \{y\} + \tilde{f}^i$, for $i = 1, \dots, l$. The adjacency lists are $F_{v^0} = \{f^1, \dots, f^l\}$, $F_{z^i} = \{f^j |\ \tilde{f}^j \in \tilde{F}_{z^i}\}$ and $V_{f^i} = \{v^0\} \cup \{z^j |\ z^j \in \tilde{V}_{\tilde{f}^i}\}$ for $i = 1, \dots, l$.

Variant 2. Substitution of line 5 of Algorithm 1.

```
if $i = 0$ then
    Initialize $P^0$ as described (with vertices $V^0$ and facets $F^0$ and adjacency lists);
else
    $P^i = onlinevert2(P^{i-1}, H)$;
end if
```

5. Computational tests

There is a MATLAB implementation of Algorithm 1, which uses the vertex enumeration procedure (*vert*) that has been used also in [22]. The procedures explained in Section 3 as well as the variants of Algorithm 1 given in Section 4 are implemented using MATLAB.

In order to compare the performances of the vertex enumeration procedure of Algorithm 1, Variants 1, and 2, we randomly generate linear multiobjective optimization problems in the following form:

$$\text{minimize} \quad Cx \quad \text{subject to} \quad Ax \leq b, \ \ x \geq 0,$$

where $C \in \mathbb{R}^{d \times n}$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and the ordering cone is $\mathbb{R}^d_+$. As the objective space is d-dimensional, the vertex enumeration problem to be solved is also d-dimensional, see Algorithm 1 line 5. For our tests, each component of A and C is generated using independent normal distributions with mean $\mu = 0$ and variance $\sigma^2 = 100$, whereas each component of vector b is generated using independent uniform distributions on range $[0, 10]$. In order to avoid numerical complications, we round each component of A, C and b to its closest integer. When we generate a problem, we first check the feasibility and boundedness (in the sense that the ideal point is finite) of it and add it to our sample only if the problem is bounded and feasible, hence solvable. Otherwise, we continue generating another set of C, A and b.

We generate different set of problems where the number of objectives (d) ranges from 2 to 4; the number of constraints (m) is taken as $2n$, where n is the number of variables. For the problems with two objectives ($d = 2$), we generate 30 feasible and bounded linear MOPs and for $d = 3$ and $d = 4$, we generate 20 of them.

The aim of the computational tests is to compare the performances of different vertex enumeration procedures that is called in each iteration of Algorithm 1 (respectively Variants 1 and 2). Note that the efficiencies of Algorithm 1 and the two variants depend also on the choice of the vertex to be considered in each iteration, see line 7 of Algorithm 1. Indeed, for the current implementation of Algorithm 1, a vertex v is chosen arbitrarily (depending on the structure of the list of vertices to be considered). Hence, calling Algorithm 1, Variants 1 and 2 separately for the same test problem and checking the overall performances does not necessarily yield a fair

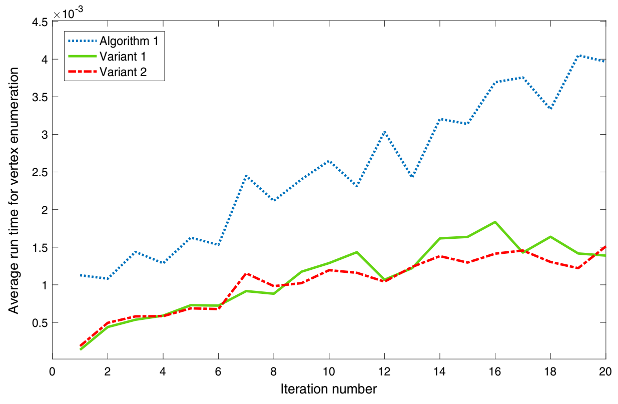

FIGURE 1. Run time performances of vertex enumeration procedures for problems with $d = 2$ and $n = 50$. The total number of iterations is 20; and the sample size is 20.

comparison of the performances of the vertex enumeration procedures that is used within. It is possible that the algorithm and the variants go over the vertices of the current outer approximation in different orders. Hence, starting from the earliest iterations, each variant may yield a different current outer approximation, which of course affect the overall performance.

In order to overcome this difficulty, we solve the problems using Algorithm 1, but in each iteration we solve the same vertex enumeration problem using three different methods: *vert* from [22], *onlinevert* and *onlinevert*2. In order to do that we had three different initialization for each procedure. We set $M = 10^4$ for Variant 1. We measure the CPU times that is spent during each vertex enumeration procedure starting from the first iteration. The approximation error ϵ is taken as 0.005 for the bi-objective problems and as 0.05 for the problems with more than two objectives.

The tests are conducted on a computer with system features Intel(R) Core(TM) i5-7200U CPU@ 2.50 GHz 2.71 GHz, 8.00 GB RAM, X64 Windows 10 and we utilize MATLAB R2013a.

We compare the average CPU times that are spent during these vertex enumeration procedures in each iteration of Algorithm 1. Indeed, we consider a sub-sample of problems: To explain it with an example, for $d = 2$, $n = 50$, we generate 30 problems among which the minimum number of iterations (of Algorithm 1) required is observed as 3. If we want to have a sample of 30 instances requiring the same number of iterations, we need to consider only the first 3 iterations of all these problems. Instead of considering 30 rather small-sized (3 iterations) problems, we reduce the sample size to 20 and we increase the number of iterations accordingly. More precisely, we list 30 problems according to the number of iterations that they require in a non-increasing order. Then, we consider the first 20 problems. Figure 1 shows the average run time spent for the vertex enumeration in each iteration. We observe that the run time of the vertex enumeration procedure used in Algorithm 1 is slightly more than the twice of the time spent by the Variants 1, and 2. However, there is no clear distinction between the two variants for these instances.

For $d = 3$, we consider four set of parameters, where we take the number of variables n as 5, 10, 20, 30. Here, we expect that the MOPs would require more iterations as the size of the problem increases. The motivation of generating different sizes is to observe this pattern and if this is the case, then to observe the performance of the different vertex enumeration procedures as the iteration number increases. For each set, we generate 20 problems and consider the sub-samples of sizes 15, as explained before. The graphs can be seen in Figure 2.

As the number of variables of the MOP increases, we observe that the number of iterations required for the algorithm increases, as expected. Moreover, we see that the average CPU time spent for each iteration increases as the iteration number increases. This is expected since, in general, the number of vertices to be checked in each iteration increases.

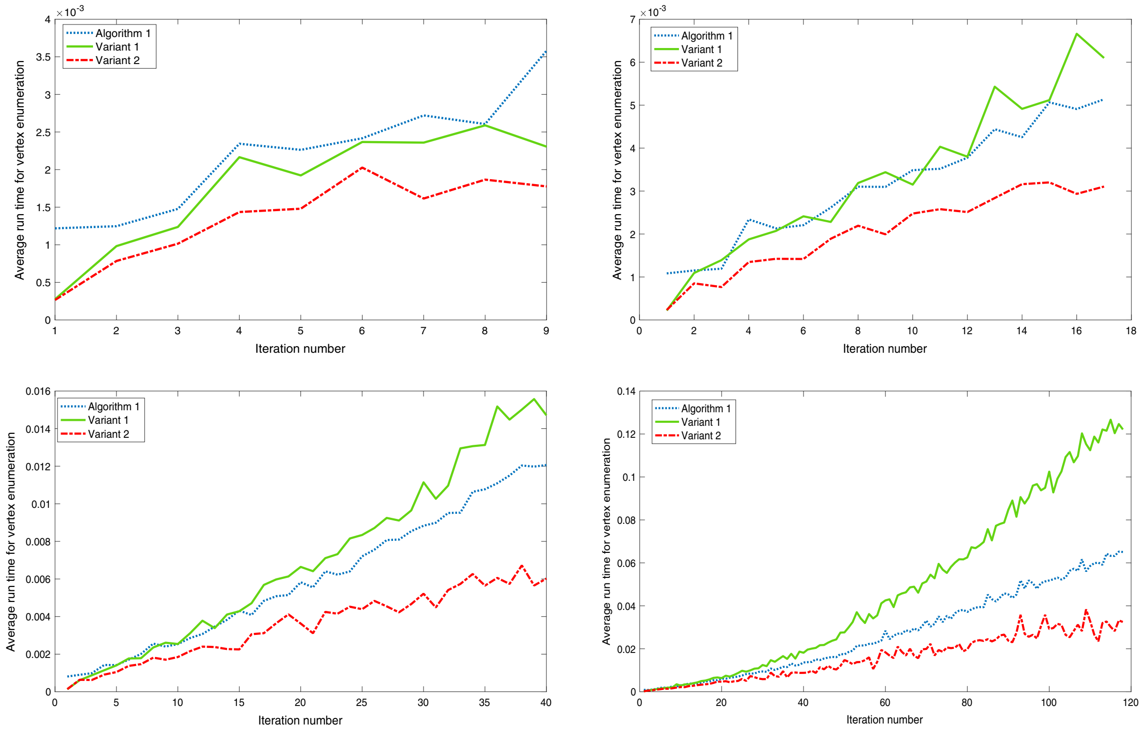

FIGURE 2. Run time performances of vertex enumeration procedures for problems with $d = 3$ and $n = 5$ (*top left*), $n = 10$ (*top right*), $n = 20$ (*bottom left*), $n = 30$ (*bottom right*). The total number of iterations are 9, 17, 40, 118, respectively; and the sample size is 15 for all.

In the first two graphs ($n = 5$ and $n = 10$) of Figure 2, the performances of Algorithm 1 and Variant 1 are similar, whereas Variant 2 seems to work faster then both. As the iteration number increases, which is the case for larger problems ($n = 20$ and $n = 30$), the differences in the performances also increase. Moreover, it is observed that Variant 1 gets worse than Algorithm 1 as the iteration number increases.

For $d = 4$, we generate 20 problems with $n = 5$ and we consider two sub-samples of sizes 15 and 10. The graphs can be seen in Figure 3. We observe a similar pattern as we observed for three dimensional problems. Different from those, the pattern is clear even from the earliest iterations.

By checking Figures 1–3, we observe that the time required for vertex enumeration increases as the dimension of the (objective) space d increases. For example, if one considers the average CPU time spent in 20th iteration for $d = 2, d = 3$ (check for instance bottom left figure with $n = 20$) and $d = 4$, these are respectively around $0.004, 0.006$ and 0.01 for Algorithm 1; $0.0015, 0.007$ and 0.015 for Variant 1; and $0.0015, 0.003$ and 0.004 for Variant 2. Indeed, we see that the increase in the run times is the most for Variant 1.

Note that *vert* from [22] solves an off-line VEP whereas *onlinevert* solves an on-line VEP in each iteration. Hence, it may not be expected to observe that *vert* (Algorithm 1) is more efficient than *onlinevert* (Variant 1), especially as the number of iterations increases. This occurred for higher dimensional problems possibly because there are too many artificial vertices to be considered even though they are deleted at the very end of the algorithm, see Variant 1 – Substitution of line 17 of Algorithm 1. For two sets of random instances (15 instances with $d = 3, n = 20$ and 10 instances with $d = 4, n = 5$), we check the number of artificial and actual vertices that are generated at each iteration. Among these random instances, the minimum numbers of iterations required are 45 for the set with $d = 3$; and 17 for the set with $d = 4$. Hence, we check the first 45 and respectively,

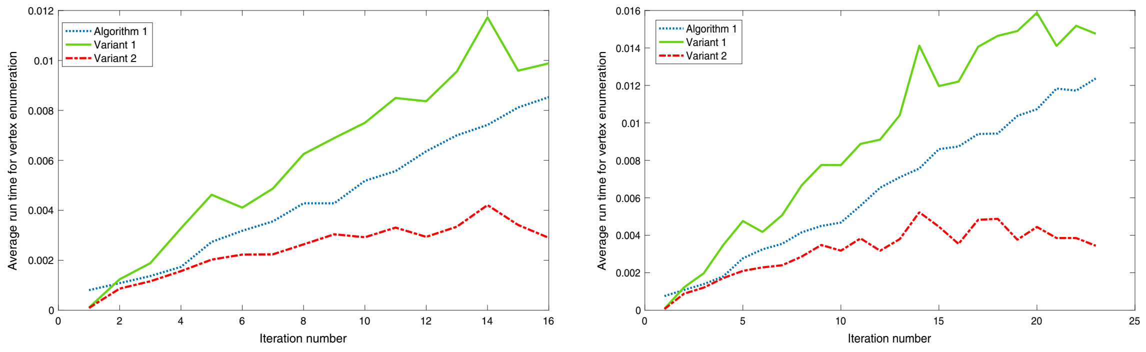

Figure 3. Run time performances of vertex enumeration procedures for problems with $d = 4$, $n = 5$. The total number of iterations are 16 and 23; and the sample size is 15 and 10, respectively (*left* and *right*).

Table 1. Number of actual and artificial vertices for test instances with $d = 3$, $n = 20$ and with $d = 4, n = 5$. The sample size is 15 for $d = 3$ instances (* except for the last three rows, for which the sample size is 10); and it is 10 for $d = 4$ instances (** except the last three rows, for which the sample sizes are 9, 8 and 6, respectively).

$d = 3, n = 20$				$d = 4, n = 5$			
# iteration	# actual vertices	# artificial vertices	% artificial	# iteration	# actual vertices	# artificial vertices	% artificial
10	14.00	7.27	34.17	5	4.80	7.30	60.33
20	27.27	10.80	28.37	10	12.10	17.60	59.26
30	41.60	13.20	24.09	13	14.40	21.00	59.32
40	54.13	16.47	23.32	17	15.70	25.20	61.61
50^*	73.00	18.10	19.87	21^{**}	17.22	28.89	62.65
60^*	86.70	20.90	19.42	23^{**}	16.87	29.63	63,71
70^*	100.90	24.30	19.41	26^{**}	17.00	32.50	65.66

17 iterations of corresponding sets of instances. Each row in Table 1 shows for the particular iteration, the average number of actual and artificial vertices as well as the percentage of the artificial vertices within all. Note that instead of providing this information for every iteration, we pick some of them as this is sufficient to summarize the general trend.

For the problems with $d = 3$, we observe that on the average, 20 percent of the vertices considered for *onlinevert* in each iteration are artificial. This percentage is higher for the earlier iterations and decreases later on. However, the average number of artificial vertices are increasing as in general the number of vertices increases rapidly through iterations. Indeed, for a sub-sample of size 10, we can increase the iteration number up to 70 and we observe the same pattern, see the last three rows of Table 1.

For the problems with $d = 4$, more than half of the vertices considered for *onlinevert* are observed to be artificial. Different from the previous case, this percentage is increasing through the iterations. Note that we can increase the iteration number up to 26 by decreasing the sample size (see the last three rows of Tab. 1) and the same pattern holds even then. This explains the poor performance of Variant 1 for high dimensional problems.

6. Conclusion

We study the vertex enumeration problem, in particular the DD method (*onlinevert*) to be used within an objective space based VOP algorithm (Algorithm 1), which currently employs an offline vertex enumeration procedure *vert*. We also consider a modified DD method (*onlinevert2*) which works for unbounded polyhedrons with known recession cones, as proposed in [8]. We provide a detailed Euclidean space description of this method. We examine two variants of Algorithm 1, one using *onlinevert* and the other using *onlinevert*2. We compare the performances through randomly generated linear MOP instances.

Overall, *onlinevert2* used in Variant 2 is observed to be the most efficient procedure among the others especially as the dimension of the objective space and also as the number of iterations increases. We demonstrate through a detailed computational study that, for vertex enumeration problems, where the polyhedron to be computed is unbounded with a known recession cone, employing the modified variant of the DD method (*onlinevert2*) increases the overall efficiency significantly. As discussed throughout, one crucial application area is the objective space based MOP algorithms. However, it could be employed as a subroutine for any algorithm or procedure that requires solving VEPs with the aforementioned property.

Acknowledgements. We would like to thank two anonymous referees for helpful comments and pointing us to further related literature.

References

[1] W. Altherr, An algorithm for enumerating all vertices of a convex polyhedron. *Computing* **15** (1975) 181–193.

[2] D. Avis, Computational experience with the reverse search vertex enumeration algorithm. *Optim. Methods Softw.* **10** (1998) 107–124.

[3] D. Avis and K. Fukuda, A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete Comput. Geom.* **8** (1992) 295–313.

[4] D. Avis, D. Bremner and R. Seidel, How good are convex hull algorithms? *Comput. Geom. Theory App.* **7** (1997) 265–302.

[5] M.L. Balinski, An algorithm for finding all vertices of a convex polyhedral set. *J. Soc. Ind. Appl. Math.* **9** (1961) 72–88.

[6] C. Barber, D. Dobkin and H. Huhdanpaa, The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw. ACM TOMS* **22** (1996) 469–483.

[7] H.P. Benson, An outer approximation algorithm for generating all efficient extreme points in the outcome set of a multiple objective linear programming problem. *J. Global Optim.* **13** (1998) 1–24.

[8] B.A. Burton and M. Ozlen, Projective geometry and the outer approximation algorithm for multiobjective linear programming. Preprint arXiv:1006.3085 (2010).

[9] P.C. Chen, P. Hansen and B. Jaumard, On-line and off-line vertex enumeration by adjacency lists. *Oper. Res. Lett.* **10** (1991) 403–409.

[10] W.D. Cook and L.M. Seiford, Data envelopment analysis (dea)–thirty years on. *Eur. J. Oper. Res.* **192** (2009) 1–17.

[11] W.W. Cooper, L.M. Seiford and J. Zhu, Handbook on Data Envelopment Analysis. Vol. 164 of : *International Series in Operations Research & Management Science*. Springer, New York, NY (2011).

[12] L. Csirmaz, Using multiobjective optimization to map the entropy region. *Comput. Optim. App.* **63** (2016) 45–67.

[13] M.E. Dyer and L.G. Proll, An improved vertex enumeration algorithm. *Eur. J. Oper. Res.* **9** (1982) 359–368.

[14] M. Ehrgott, Multicriteria Optimization. Springer, New York, NY (2005).

[15] M. Ehrgott, L. Shao and A. Schöbel, An approximation algorithm for convex multi-objective programming problems. *J. Global Optim.* **50** (2011) 397–416.

[16] M. Ehrgott, M. Hasannasab and A. Raith, A multiobjective optimization approach to compute the efficient frontier in data envelopment analysis. *J. Multi-Criteria Decis. Anal.* **26** (2019) 187–198.

[17] K. Fukuda and A. Prodon, Double description method revisited. *Comb. Comput. Sci.* **1120** (1996) 91–111.

[18] A.H. Hamel, A. Löhne and B. Rudloff, Benson type algorithms for linear vector optimization and applications. *J. Global Optim.* **59** (2014) 811–836.

[19] L. Khachiyan, E. Boros, K. Borys, K. Elbassioni and V. Gurvich, Generating all vertices of a polyhedron is hard. *Discrete Comput. Geom.* **39** (2008) 174–190.

[20] J.S. Liu, L.Y.Y. Lu, W. Lu and B.J.Y. Lin, A survey of DEA applications. *Omega* **41** (2013) 893–902.

[21] A. Löhne, Vector Optimization with Infimum and Supremum. Springer, Berlin-Heidelberg (2011).

[22] A. Löhne, BENSOLVE: A free VLP solver, version 1.2 (2012).

[23] A. Löhne and B. Weißing, BENSOLVE: A free VLP solver, version 2.0.1 (2015).

[24] A. Löhne, B. Rudloff and F. Ulus, Primal and dual approximation algorithms for convex vector optimization problems. *J. Global Optim.* **60** (2014) 713–736.

[25] S.T. Motzkin, G.L. Raiffa, G.L. Thompson and M.L. Thrall, The double description method. *Contrib. Theory Games* **2** (1953) 51–74.

[26] A. Pascoletti and P. Serafini, Scalarizing vector optimization problems. *J. Optim. Theory App.* **42** (1984) 499–524.

[27] R.T. Rockafellar, Convex Analysis. Princeton University Press (1970).

[28] B.K. Schmidt and T.H. Mattheiss, Computational results on an algorithm for finding all vertices of a poltyope. *Math. Program.* **18** (1980) 308–329.

[29] H. Tuy, Canonical DC programming problem: outer approximation methods revisited. *Oper. Res. Lett.* **18** (1995) 99–106.

[30] P. Zhou, B.W. Ang and K.L. Poh, A survey of data envelopment analysis in energy and environmental studies. *Eur. J. Oper. Res.* **189** (2008) 1–18.