Math. Program., Ser. A (2017) 163:213–242
DOI 10.1007/s10107-016-1061-z

FULL LENGTH PAPER

A parametric simplex algorithm for linear vector optimization problems

Birgit Rudloff[1] · Firdevs Ulus[2] · Robert Vanderbei[3]

Received: 9 July 2015 / Accepted: 29 July 2016 / Published online: 8 August 2016

Abstract In this paper, a parametric simplex algorithm for solving linear vector optimization problems (LVOPs) is presented. This algorithm can be seen as a variant of the multi-objective simplex (the Evans–Steuer) algorithm (Math Program 5(1):54–72, 1973). Different from it, the proposed algorithm works in the parameter space and does not aim to find the set of all efficient solutions. Instead, it finds a solution in the sense of Löhne (Vector optimization with infimum and supremum. Springer, Berlin, 2011), that is, it finds a subset of efficient solutions that allows to generate the whole efficient frontier. In that sense, it can also be seen as a generalization of the parametric self-dual simplex algorithm, which originally is designed for solving single objective linear optimization problems, and is modified to solve two objective bounded LVOPs with the positive orthant as the ordering cone in Ruszczyński and Vanderbei (Econometrica 71(4):1287–1297, 2003). The algorithm proposed here works for any dimension, any solid pointed polyhedral ordering cone C and for bounded as well as unbounded problems. Numerical results are provided to compare the proposed algorithm with an objective space based LVOP algorithm [Benson's algorithm in Hamel et

✉ Firdevs Ulus
firdevs@bilkent.edu.tr

Birgit Rudloff
birgit.rudloff@wu.ac.at

Robert Vanderbei
rvdb@princeton.edu

1 Institute for Statistics and Mathematics, Vienna University of Economics and Business, 1020 Vienna, Austria

2 Department of Industrial Engineering, Bilkent University, 06800 Ankara, Turkey

3 Department of Operations Research and Financial Engineering, Princeton University, Princeton, NJ 08544, USA

al. (J Global Optim 59(4):811–836, 2014)], that also provides a solution in the sense of Löhne (2011), and with the Evans–Steuer algorithm (1973). The results show that for non-degenerate problems the proposed algorithm outperforms Benson's algorithm and is on par with the Evans–Steuer algorithm. For highly degenerate problems Benson's algorithm (Hamel et al. 2014) outperforms the simplex-type algorithms; however, the parametric simplex algorithm is for these problems computationally much more efficient than the Evans–Steuer algorithm.

Keywords Linear vector optimization · Multiple objective optimization · Algorithms · Parameter space segmentation

Mathematics Subject Classification 90C29 · 90C05 · 90-08

1 Introduction

Vector optimization problems have been studied for decades and many methods have been developed to solve or approximately solve them. In particular, there are a variety of algorithms to solve linear vector optimization problems (LVOPs).

1.1 Related literature

Among the algorithms that can solve LVOPs, some are extensions of the simplex method and are working in the variable space. In 1973, Evans and Steuer [15] developed a multi-objective simplex algorithm that finds the set of all 'efficient extreme solutions' and the set of all 'unbounded efficient edges' in the variable space, see also [11, Algorithm 7.1]. Later, some variants of this algorithm have been developed, see for instance [1,2,9,10,17,33]. More recently, Ehrgott, Puerto and Rodriguez-Chía [13] developed a primal–dual simplex method that works in the parameter space. This algorithm does not guarantee to find the set of all efficient solutions, instead it provides a subset of efficient solutions that are enough to generate the whole efficient frontier in case the problem is 'bounded'. All of these simplex-type algorithms are designed to solve LVOPs with any number of objective functions where the ordering is componentwise. Among these, the Evans–Steuer algorithm [15] is implemented as a software called ADBASE [31]. The idea of decomposing the parameter set is also used to solve multiobjective integer programs, see for instance [26].

In [27], Ruszczyński and Vanderbei developed an algorithm to solve LVOPs with two objectives and the efficiency of this algorithm is equivalent to solving a single scalar linear program by the parametric simplex algorithm. Indeed, the algorithm is a modification of the parametric simplex method and it produces a subset of efficient solutions that generate the whole efficient frontier in case the problem is bounded.

Apart from the algorithms that work in the variable or parameter space, there are algorithms working in the objective space. In [8], Dauer and Liu proposed a procedure to determine the 'maximal' extreme points and edges of the image of the feasible region. Later, Benson [5] proposed an outer approximation algorithm that also works in the objective space. These methods are motivated by the observation that the dimension of the objective space is usually much smaller than the dimension of the variable space, and decision makers tend to choose a solution based on objective values rather than variable values, see for instance [7]. Löhne [19] introduced a solution concept for LVOPs that takes into account these ideas. Accordingly a solution consists of a set of 'point maximizers (efficient solutions)' and a set of 'direction maximizers (unbounded efficient edges)', which altogether generate the whole efficient frontier. If a problem is 'unbounded', then a solution needs to have a nonempty set of direction maximizers. There are several variants of Benson's algorithm for LVOPs. Some of them can also solve unbounded problems as long as the image has at least one vertex, but only by using an additional Phase 1 algorithm, see for instance [19, Section 5.4]. The algorithms provided in [12,19,29,30] solve in each iteration at least two LPs that are of the same size as the original problem. An improved variant where only one LP has to be solved in each iteration has been proposed independently in [6] and [16]. In addition to solving (at least) one LP, these algorithms solve a vertex enumeration problem in each iteration. As it can be seen in [6,22,23,29], it is also possible to employ an online vertex enumeration method. In this case, instead of solving a vertex enumeration problem from scratch in each iteration, the vertices are updated after an addition of a new inequality. Recently, Benson's algorithm was extended to approximately solve bounded convex vector optimization problems in [14,21].

1.2 The proposed algorithm

In this paper, we develop a parametric simplex algorithm to solve LVOPs of any size and with any solid pointed polyhedral ordering cone. Although the structure of the algorithm is similar to the Evans–Steuer algorithm, it is different since the algorithm proposed here works in the parameter space and it finds a solution in the sense that Löhne proposed in [19]. In other words, instead of generating the set of all point and direction maximizers, it only finds a subset of them that already allows to generate the whole efficient frontier. More specifically, the difference can be seen at two points. First, in each iteration instead of performing a pivot for each 'efficient nonbasic variable', we perform a pivot only for a subset of them. This already decreases the total number of pivots performed throughout the algorithm. In addition, the method of finding this subset of efficient nonbasic variables is more efficient than the method that is needed to find the whole set. Secondly, for an entering variable, instead of performing all possible pivots for all 'efficient basic variables' as in [15], we perform a single pivot by picking only one of them as the leaving variable. In this sense, the algorithm provided here can also be seen as a generalization of the algorithm proposed by Ruszczyǹski and Vanderbei [27] to unbounded LVOPs with more than two objectives and with more general ordering cones.

In each iteration the algorithm provides a set of parameters which make the current vertex optimal. This parameter set is given by a set of inequalities among which the redundant ones are eliminated. This is an easier procedure than the vertex enumeration problem, which is required in some objective space algorithms. Different from the objective space algorithms, the algorithm provided here does not require to solve an additional LP in each iteration. Moreover, the parametric simplex algorithm works also for unbounded problems even if the image has no vertices and generates direction maximizers at no additional cost.

As in the scalar case, the efficiency of simplex-type algorithms is expected to be better whenever the problem is non-degenerate. In vector optimization problems, one may observe different types of redundancies if the problem is degenerate. The first one corresponds to the ‘primal degeneracy’ concept in scalar problems. In this case, a simplex-type algorithm may find the same ‘efficient solution’ for many iterations. That is to say, one remains at the same vertex of the feasible region for more than one iteration. The second type of redundancy corresponds to the ‘dual degeneracy’ concept in scalar problems. Accordingly, the algorithm may find different ‘efficient solutions’ which yield the same objective values. In other words, one remains at the same vertex of the image of the feasible region. Additionally to these, a simplex-type algorithm for LVOPs may find efficient solutions which yield objective values that are not vertices of the image of the feasible region. Note that these points that are on a non-vertex face of the image set are not necessary to generate the whole efficient frontier. Thus, one can consider these solutions also as redundant.

The parametric simplex algorithm provided here may also find redundant solutions. However, it will be shown that the algorithm terminates at a finite time, that is, there is no risk of cycling. Moreover, compared to the Evans–Steuer algorithm, the parametric simplex algorithm finds much fewer redundant solutions in general.

We provide different initialization methods. One of the methods requires to solve two LPs while a second method can be seen as a Phase 1 algorithm. Both of these methods work for any LVOP. Depending on the structure of the problem, it is also possible to initialize the algorithm without solving an LP or performing a Phase 1 algorithm.

This paper is structured as follows. Section 2 is dedicated to basic concepts and notation. In Sect. 3, the linear vector optimization problem and solution concepts are introduced. The parametric simplex algorithm is provided in Sect. 4. Different methods of initialization are explained in Sect. 4.8. Illustrative examples are given in Sect. 5. In Sect. 6, we compare the parametric simplex algorithm provided here with the different simplex algorithms for solving LVOPs that are available in the literature. Finally, some numerical results regarding the efficiency of the proposed algorithm compared to Benson’s algorithm and the Evans–Steuer algorithm are provided in Sect. 7.

2 Preliminaries

For a set $A \subseteq \mathbb{R}^q$, A^C, $\operatorname{int} A$, $\operatorname{ri} A$, $\operatorname{cl} A$, $\operatorname{bd} A$, $\operatorname{conv} A$, $\operatorname{cone} A$ denote the complement, interior, relative interior, closure, boundary, convex hull, and conic hull of it, respectively. If $A \subseteq \mathbb{R}^q$ is a non-empty polyhedral convex set, it can be represented as

$$A = \operatorname{conv}\{x^1, \dots, x^s\} + \operatorname{cone}\{k^1, \dots, k^t\}, \tag{1}$$

where $s \in \mathbb{N}\backslash\{0\}, t \in \mathbb{N}$, each $x^i \in \mathbb{R}^q$ is a point, and each $k^j \in \mathbb{R}^q\backslash\{0\}$ is a *direction* of A. Note that $k \in \mathbb{R}^q\backslash\{0\}$ is called a direction of A if $A + \{\alpha k \in \mathbb{R}^q \mid \alpha >$

$0\} \subseteq A$. The set $A_\infty := \operatorname{cone}\{k^1, \dots, k^t\}$ is the recession cone of A. The set of points $\{x^1, \dots, x^s\}$ together with the set of directions $\{k^1, \dots, k^t\}$ are called the *generators* of the polyhedral convex set A. We say $(\{x^1, \dots, x^s\}, \{k^1, \dots, k^t\})$ is a V-representation of A whenever (1) holds. For convenience, we define $\operatorname{cone}\emptyset = \{0\}$.

A convex cone C is said to be *non-trivial* if $\{0\} \subsetneq C \subsetneq \mathbb{R}^q$ and *pointed* if it does not contain any line. A non-trivial convex pointed cone C defines a partial ordering $\leq_C$ on $\mathbb{R}^q$:

$$v \leq_C w \;:\Leftrightarrow\; w - v \in C.$$

For a non-trivial convex pointed cone $C \subseteq \mathbb{R}^q$, a point $y \in A$ is called a *C-maximal element* of A if $(\{y\} + C\backslash\{0\}) \cap A = \emptyset$. If the cone C is *solid*, that is, if it has a non-empty interior, then a point $y \in A$ is called *weakly C-maximal* if $(\{y\} + \operatorname{int} C) \cap A = \emptyset$. The set of all (weakly) C-maximal elements of A is denoted by $(\mathrm{w})\mathrm{Max}_C\,(A)$. The set of (weakly) C-minimal elements is defined by $(\mathrm{w})\mathrm{Min}_C\,(A) := (\mathrm{w})\mathrm{Max}_{-C}\,(A)$. The (positive) dual cone of C is the set $C^+ := \left\{z \in \mathbb{R}^q \mid \forall y \in C : z^T y \geq 0\right\}$. The positive orthant of $\mathbb{R}^q$ is denoted by $\mathbb{R}^q_+$, that is, $\mathbb{R}^q_+ := \{y \in \mathbb{R}^q \mid y_i \geq 0, i = 1, \dots, q\}$.

3 Linear vector optimization problems

We consider a linear vector optimization problem (LVOP) in the following form

$$\begin{aligned} \text{maximize} \quad & P^T x \quad (\text{with respect to } \leq_C) \\ \text{subject to} \quad & Ax \leq b, \\ & x \geq 0, \end{aligned} \tag{P}$$

where $P \in \mathbb{R}^{n\times q}$, $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, and $C \subseteq \mathbb{R}^q$ is a solid polyhedral pointed ordering cone. We denote the feasible set by $\mathcal{X} := \{x \in \mathbb{R}^n \mid Ax \leq b,\ x \geq 0\}$. Throughout, we assume that (P) is feasible, i.e., $\mathcal{X} \neq \emptyset$. The image of the feasible set is defined as $P^T[\mathcal{X}] := \{P^T x \in \mathbb{R}^q \mid x \in \mathcal{X}\}$.

We consider the solution concept for LVOPs as in [19]. To do so, let us recall the following. A point $\bar{x} \in \mathcal{X}$ is said to be a *(weak) maximizer* for (P) if $P^T\bar{x}$ is (weakly) C-maximal in $P[\mathcal{X}]$. The set of (weak) maximizers of (P) is denoted by (w)Max(P). The homogeneous problem of (P) is given by

$$\begin{aligned} \text{maximize} \quad & P^T x \quad (\text{with respect to } \leq_C) \\ \text{subject to} \quad & Ax \leq 0, \\ & x \geq 0. \end{aligned} \tag{P^h}$$

The feasible region of (P^h), namely $\mathcal{X}^h := \{x \in \mathbb{R}^n \mid Ax \leq 0,\ x \geq 0\}$, satisfies $\mathcal{X}^h = \mathcal{X}_\infty$, that is, the non-zero points in $\mathcal{X}^h$ are exactly the directions of $\mathcal{X}$. A direction $k \in \mathbb{R}^n\backslash\{0\}$ of $\mathcal{X}$ is called a *(weak) maximizer* for (P) if the corresponding point $k \in \mathcal{X}^h\backslash\{0\}$ is a (weak) maximizer of the homogeneous problem (P^h).

Definition 3.1 [16,19] A set $\bar{\mathcal{X}} \subseteq \mathcal{X}$ is called a *set of feasible points* for (P) and a set $\bar{\mathcal{X}}^h \subseteq \mathcal{X}^h\backslash\{0\}$ is called a *set of feasible directions* for (P).

A pair of sets $\left(\bar{\mathcal{X}}, \bar{\mathcal{X}}^h\right)$ is called a *finite supremizer* for (P) if $\bar{\mathcal{X}}$ is a non-empty finite set of feasible points for (P), $\bar{\mathcal{X}}^h$ is a (not necessarily non-empty) finite set of feasible directions for (P), and

$$\operatorname{conv} P^T[\bar{\mathcal{X}}] + \operatorname{cone} P^T[\bar{\mathcal{X}}^h] - C = P^T[\mathcal{X}] - C. \tag{2}$$

A finite supremizer $(\bar{\mathcal{X}}, \bar{\mathcal{X}}^h)$ of (P) is called a *solution* to (P) if it consists of only maximizers.

The set $\mathcal{P} := P^T[\mathcal{X}] - C$ is called the *lower image* of (P). Let $y^1, \dots, y^t$ be the generating vectors of the ordering cone C. Then, $(\{0\}, \{y^1, \dots, y^t\})$ is a V-representation of the cone C, that is, $C = \operatorname{cone}\{y^1, \dots, y^t\}$. Clearly, if $(\bar{\mathcal{X}}, \bar{\mathcal{X}}^h)$ is a finite supremizer, then $(P^T[\bar{\mathcal{X}}], P^T[\bar{\mathcal{X}}^h] \cup \{-y^1, \dots, -y^t\})$ is a V-representation of the lower image $\mathcal{P}$.

Definition 3.2 (P) is said to be *bounded* if there exists $p \in \mathbb{R}^q$ such that $\mathcal{P} \subseteq \{p\} - C$.

Remark 3.3 Note that the recession cone of the lower image, $\mathcal{P}_\infty$, is equal to the lower image of the homogeneous problem, that is, $\mathcal{P}_\infty = P^T[\mathcal{X}^h] - C$, see [19, Lemma 4.61]. Clearly, $\mathcal{P}_\infty \supseteq -C$, which also implies $\mathcal{P}_\infty^+ \subseteq -C^+$. In particular, if (P) is bounded, then we have $\mathcal{P}_\infty = -C$ and $\bar{\mathcal{X}}^h = \emptyset$.

The *weighted sum* scalarized problem for a parameter vector $w \in C^+$ is

$$\begin{aligned} \text{maximize} \quad & w^T P^T x \\ \text{subject to} \quad & Ax \leq b, \\ & x \geq 0, \end{aligned} \tag{P$_1(w)$}$$

and the following well known proposition holds.

Proposition 3.4 ([24, Theorem 2.5]) *A point $\bar{x} \in \mathcal{X}$ is a maximizer (weak maximizer) of (P) if and only if it is an optimal solution to $(P_1(w))$ for some $w \in \operatorname{int} C^+$ ($w \in C^+ \setminus \{0\}$).*

Proposition 3.4 suggests that if one could generate optimal solutions, whenever they exist, to the problems $(\mathrm{P}_1(w))$ for $w \in \operatorname{int} C^+$, then this set of optimal solutions $\bar{\mathcal{X}}$ would be a set of (point) maximizers of (P). Indeed, it will be enough to solve problem $(\mathrm{P}_1(w))$ for $w \in \operatorname{ri} W$, where

$$W := \{w \in C^+ |\; w^T c = 1\}, \tag{3}$$

for some fixed $c \in \operatorname{int} C$. Note that $(\mathrm{P}_1(w))$ is not necessarily bounded for all $w \in \operatorname{ri} W$. Denote the set of all $w \in \operatorname{ri} W$ such that $(\mathrm{P}_1(w))$ has an optimal solution by W_b. If one can find a finite partition $(W_b^i)_{i=1}^s$ of W_b such that for each $i \in \{1, \dots, s\}$ there exists $x^i \in \mathcal{X}$ which is an optimal solution to $(\mathrm{P}_1(w))$ for all $w \in W_b^i$, then, clearly, $\bar{\mathcal{X}} = \{x^1, \dots, x^s\}$ will satisfy (2) provided one can also generate a finite set of (direction) maximizers $\bar{\mathcal{X}}^h$. Trivially, if problem (P) is bounded, then $(\mathrm{P}_1(w))$ can be solved optimally for all $w \in C^+$, $\bar{\mathcal{X}}^h = \emptyset$, and $(\bar{\mathcal{X}}, \emptyset)$ satisfies (2). If problem (P) is unbounded, we will construct in Sect. 4 a set $\bar{\mathcal{X}}^h$ by adding certain directions to it whenever one encounters a set of weight vectors $w \in C^+$ for which $(\mathrm{P}_1(w))$ cannot be solved optimally. The following proposition will be used to prove that this set $\bar{\mathcal{X}}^h$, together with $\bar{\mathcal{X}} = \{x^1, \dots, x^s\}$ will indeed satisfy (2). It provides a characterization of the recession cone of the lower image in terms of the weighted sum scalarized problems. More precisely, the negative of the dual of the recession cone of the lower image can be shown to consist of those $w \in C^+$ for which $(\mathrm{P}_1(w))$ can be optimally solved.

Proposition 3.5 *The recession cone $\mathcal{P}_\infty$ of the lower image satisfies*

$$-\mathcal{P}_\infty^+ = \{w \in C^+ |\ (P_1(w)) \textit{ is bounded}\}.$$

Proof By Remark 3.3, we have $\mathcal{P}_\infty = P^T[\mathcal{X}^h] - C$. Using $0 \in \mathcal{X}^h$ and $0 \in C$, we obtain

$$\begin{aligned} -\mathcal{P}_\infty^+ &= \{w \in \mathbb{R}^q |\ \forall x^h \in \mathcal{X}^h, \forall c \in C : w^T(P^T x^h - c) \leq 0\} \\ &= \{w \in C^+ |\ \forall x^h \in \mathcal{X}^h : w^T P^T x^h \leq 0\}. \end{aligned} \tag{4}$$

Let $w \in -\mathcal{P}_\infty^+$, and consider the weighted sum scalarized problem of (P^h) given by

$$\begin{array}{lll} \text{maximize} & w^T P^T x & \qquad (\mathrm{P}_1^h(w)) \\ \text{subject to} & Ax \leq 0, & \\ & \ \ x \geq 0. & \end{array}$$

By (4), $x^h = 0$ is an optimal solution, which implies by strong duality of the linear program that there exist $y^* \in \mathbb{R}^m$ with $A^T y^* \geq Pw$ and $y^* \geq 0$. Then, y^* is also dual feasible for $(\mathrm{P}_1(w))$. By the weak duality theorem, $(\mathrm{P}_1(w))$ can not be unbounded.

For the reverse inclusion, let $w \in C^+$ be such that $(\mathrm{P}_1(w))$ is bounded, or equivalently, an optimal solution exists for $(\mathrm{P}_1(w))$ as we assume $\mathcal{X} \neq \emptyset$. By strong duality, the dual problem of $(\mathrm{P}_1(w))$ has an optimal solution y^*, which is also dual feasible for $(\mathrm{P}_1^h(w))$. By weak duality, $(\mathrm{P}_1^h(w))$ is bounded and has an optimal solution $\tilde{x}^h$. Then, $w \in -\mathcal{P}_\infty^+$ holds. Indeed, assuming the contrary, one can easily find a contradiction to the optimality of $\tilde{x}^h$. □

4 The parametric simplex algorithm for LVOPs

In [15], Evans and Steuer proposed a simplex algorithm to solve linear multiobjective optimization problems. The algorithm moves from one vertex of the feasible region to another until it finds the set of all extreme (point and direction) maximizers. In this paper we propose a parametric simplex algorithm to solve LVOPs where the structure of the algorithm is similar to the Evans–Steuer algorithm. Different from it, the parametric simplex algorithm provides a solution in the sense of Definition 3.1, that is, it finds subsets of extreme point and direction maximizers that generate the lower image. This allows the algorithm to deal with the degenerate problems more efficiently than the Evans–Steuer algorithm. More detailed comparison of the two algorithms can be seen in Sect. 6.

In [27], Ruszczyn̂ski and Vanderbei generalize the parametric self dual method, which originally is designed to solve scalar LPs [32], to solve two-objective bounded linear vector optimization problems. This is done by treating the second objective function as the auxiliary function of the parametric self dual algorithm. The algorithm provided here can be seen as a generalization of the parametric simplex algorithm from biobjective bounded LVOPs to q-objective LVOPs ($q \geq 2$) that are not necessarily bounded where we also allow for an arbitrary solid polyhedral pointed ordering cone C.

We first explain the algorithm for problems that have a solution. One can keep in mind that the methods of initialization proposed in Sect. 4.8 will verify if the problem has a solution or not.

Assumption 4.1 There exists a solution to problem (P).

This assumption is equivalent to having a nontrivial lower image $\mathcal{P}$, that is, $\emptyset \neq \mathcal{P} \subsetneq \mathbb{R}^q$. Clearly $\mathcal{P} \neq \emptyset$ implies $\mathcal{X} \neq \emptyset$, which is equivalent to our standing assumption. Moreover, by Definition 3.1 and Proposition 3.4, Assumption 4.1 implies that there exists a maximizer which guarantees that there exists some $w^0 \in \operatorname{int} C^+$ such that problem $(\mathrm{P}_1(w^0))$ has an optimal solution. In Sect. 4.8, we will propose methods to find such a w^0. It will be seen that the algorithm provided here finds a solution if there exists one.

4.1 The parameter set Λ

Throughout the algorithm we consider the scalarized problem $(\mathrm{P}_1(w))$ for $w \in W$ where W is given by (3) for some fixed $c \in \operatorname{int} C$. As W is $q-1$ dimensional, we will transform the parameter set W into a set $\Lambda \subseteq \mathbb{R}^{q-1}$. Assume without loss of generality that $c_q = 1$. Indeed, since C was assumed to be a solid cone, there exists some $c \in \operatorname{int} C$ such that either $c_q = 1$ or $c_q = -1$. For $c_q = -1$, one can consider problem (P) where C and P are replaced by $-C$ and $-P$.

Let $\tilde{c} = (c_1, \dots, c_{q-1})^T \in \mathbb{R}^{q-1}$ and define the function $w(\lambda) : \mathbb{R}^{q-1} \to \mathbb{R}^q$ and the set $\Lambda \subseteq \mathbb{R}^{q-1}$ as follows:

$$w(\lambda) := (\lambda_1, \dots, \lambda_{q-1}, 1 - \tilde{c}^T \lambda)^T,$$
$$\Lambda := \{\lambda \in \mathbb{R}^{q-1} |\ w(\lambda) \in C^+\}.$$

As we assume $c_q = 1$, $c^T w(\lambda) = 1$ holds for all $\lambda \in \Lambda$. Then, $w(\lambda) \in W$ for all $\lambda \in \Lambda$ and for any $w \in W$, $(w_1, \dots, w_{q-1})^T \in \Lambda$. Moreover, if $\lambda \in \operatorname{int} \Lambda$, then $w(\lambda) \in \operatorname{ri} W$ and if $w \in \operatorname{ri} W$, then $(w_1, \dots, w_{q-1})^T \in \operatorname{int} \Lambda$. Throughout the algorithm, we consider the parametrized problem

$$(\mathrm{P}_\lambda) := (\mathrm{P}_1(w(\lambda)))$$

for some generic $\lambda \in \mathbb{R}^{q-1}$.

4.2 Segmentation of Λ: dictionaries and their optimality region

We will use the terminology for the simplex algorithm as it is used in [32]. First, we introduce slack variables $[x_{n+1}, \dots, x_{n+m}]^T$ to obtain $x \in \mathbb{R}^{n+m}$ and rewrite (P$_\lambda$) as follows

$$\begin{aligned} \text{maximize} \quad & w(\lambda)^T [P^T \;\; 0] x \\ \text{subject to} \quad & [A \; I] x = b, \\ & x \geq 0, \end{aligned} \tag{P$_\lambda$}$$

where I is the identity and 0 is the zero matrix, all in the correct sizes. We consider a partition of the variable indices $\{1, 2, \dots, n+m\}$ into two sets $\mathcal{B}$ and $\mathcal{N}$. Variables x_j, $j \in \mathcal{B}$, are called *basic variables* and x_j, $j \in \mathcal{N}$, are called *nonbasic variables*. We write $x = \begin{bmatrix} x_{\mathcal{B}}^T & x_{\mathcal{N}}^T \end{bmatrix}^T$ and permute the columns of $[A \;\; I]$ to obtain $\begin{bmatrix} B & N \end{bmatrix}$ satisfying $[A \;\; I]x = Bx_{\mathcal{B}} + Nx_{\mathcal{N}}$, where $B \in \mathbb{R}^{m \times m}$ and $N \in \mathbb{R}^{m \times n}$. Similarly, we form matrices $P_{\mathcal{B}} \in \mathbb{R}^{m \times q}$, and $P_{\mathcal{N}} \in \mathbb{R}^{n \times q}$ such that $[P^T \;\; 0]x = P_{\mathcal{B}}^T x_{\mathcal{B}} + P_{\mathcal{N}}^T x_{\mathcal{N}}$. In order to keep the notation simple, instead of writing $[P^T \;\; 0]x$ we will occasionally write $P^T x$, where x stands then for the original decision variables in $\mathbb{R}^n$ without the slack variables.

Whenever B is nonsingular, $x_{\mathcal{B}}$ can be written in terms of the nonbasic variables as $x_{\mathcal{B}} = B^{-1}b - B^{-1}Nx_{\mathcal{N}}$. Then, the objective function of (P$_\lambda$) is

$$w(\lambda)^T [P^T \;\; 0]x = w(\lambda)^T \xi(\lambda) - w(\lambda)^T Z_{\mathcal{N}}^T x_{\mathcal{N}},$$

where $\xi(\lambda) = P_{\mathcal{B}}^T B^{-1} b$ and $Z_{\mathcal{N}} = (B^{-1}N)^T P_{\mathcal{B}} - P_{\mathcal{N}}$.

We say that each choice of basic and nonbasic variables defines a unique *dictionary*. Denote the dictionary defined by $\mathcal{B}$ and $\mathcal{N}$ by D. The *basic solution* that corresponds to D is obtained by setting $x_{\mathcal{N}} = 0$. In this case, the values of the basic variables become $B^{-1}b$. Both the dictionary and the basic solution corresponding to this dictionary are said to be *primal feasible* if $B^{-1}b \geq 0$. Moreover, if $w(\lambda)^T Z_{\mathcal{N}}^T \geq 0$, then we say that the dictionary D and the corresponding basic solution are *dual feasible*. We call a dictionary and the corresponding basic solution *optimal* if they are both primal and dual feasible.

For $j \in \mathcal{N}$, introduce the halfspace

$$I_j^D := \{\lambda \in \mathbb{R}^{q-1} | \; w(\lambda)^T Z_{\mathcal{N}}^T e^j \geq 0\},$$

where $e^j \in \mathbb{R}^n$ denotes the unit column vector with the entry corresponding to the variable x_j being 1. Note that if D is known to be primal feasible, then D is optimal for $\lambda \in \Lambda^D$, where

$$\Lambda^D := \bigcap_{j \in \mathcal{N}} I_j^D.$$

The set $\Lambda^D \cap \Lambda$ is said to be the *optimality region* of dictionary D.

Proposition 3.4 already shows that a basic solution corresponding to a dictionary D with $\Lambda^D \cap \Lambda \neq \emptyset$ yields a (weak) maximizer of (P). Throughout the algorithm we will move from dictionary to dictionary and collect their basic solutions into a set $\bar{\mathcal{X}}$. We will later show that this set will be part of the solution $(\bar{\mathcal{X}}, \bar{\mathcal{X}}^h)$ of (P). The algorithm will yield a partition of the parameter set Λ into optimality regions of dictionaries and regions where (P$_\lambda$) is unbounded. The next subsections explain how to move from one dictionary to another and how to detect and deal with unbounded problems.

4.3 The set J^D of entering variables

We call $(I_j^D)_{j \in J^D}$ a *defining (non-redundant)* collection of half-spaces of the optimality region $\Lambda^D \cap \Lambda$ if $J^D \subseteq \mathcal{N}$ satisfies

$$\Lambda^D \cap \Lambda = \bigcap_{j \in J^D} I_j^D \cap \Lambda \quad \text{and}$$
$$\Lambda^D \cap \Lambda \subsetneq \bigcap_{j \in J} I_j^D \cap \Lambda, \quad \text{for any } J \subsetneq J^D. \tag{5}$$

For a dictionary D, any nonbasic variable x_j, $j \in J^D$, is a candidate entering variable. Let us call the set J^D an *index set of entering variables* for dictionary D.

For each dictionary throughout the algorithm, an index set of entering variables is found. This can be done e.g. by the following two methods. Firstly, using the duality of polytopes, the problem of finding defining inequalities can be transformed to the problem of finding a convex hull of given points. Then, the algorithms developed for this matter, see for instance [3], can be employed. Secondly, in order to check if $j \in \mathcal{N}$ corresponds to a defining or a redundant inequality one can consider the following linear program in $\lambda \in \mathbb{R}^{q-1}$

$$\begin{aligned} &\text{maximize} && w(\lambda)^T Z_{\mathcal{N}}^T e^j \\ &\text{subject to} && w(\lambda)^T Z_{\mathcal{N}}^T e^{\bar{j}} \geq 0, \quad \text{for all } \bar{j} \in \mathcal{N} \setminus (\{j\} \cup J^{\text{redun}}), \\ & && w(\lambda)^T Y \geq 0, \end{aligned}$$

where J^{redun} is the index set of redundant inequalities that have been already found and $Y = [y^1, \dots, y^t]$ is the matrix where $y^1, \dots, y^t$ are the generating vectors of the ordering cone C. The inequality corresponding to the nonbasic variable x_j is redundant if and only if an optimal solution to this problem yields $w(\lambda^*)^T Z_{\mathcal{N}}^T e^j \leq 0$. In this case, we add j to the set J^{redun}. Otherwise, it is a defining inequality for the region $\Lambda^D \cap \Lambda$ and we add j to the set J^D. The set J^D is obtained by solving this linear program successively for each untested inequality against the remaining.

Remark 4.2 For the numerical examples provided in Sect. 5, the second method is employed. Note that the number of variables for each linear program is $q - 1$, which is much smaller than the number of variables n of the original problem in general.

Therefore, each linear program can be solved accurately and fast. Thus, this is a reliable and sufficiently efficient method to find J^D.

Before applying one of these methods, one can also employ a modified Fourier–Motzkin elimination algorithm as described in [4] in order to decrease the number of redundant inequalities. Note that this algorithm has a worst-case complexity of $\mathcal{O}(2^{q-1}(q-1)^2)n^2)$. Even though it does not guarantee to detect all of the redundant inequalities, it decreases the number significantly.

Note that different methods may yield a different collection of indices as the set J^D might not be uniquely defined. However, the proposed algorithm works with any choice of J^D.

4.4 Pivoting

In order to initialize the algorithm one needs to find a dictionary D^0 for the parametrized problem (P$_\lambda$) such that the optimality region of D^0 satisfies $\Lambda^{D^0} \cap \operatorname{int} \Lambda \neq \emptyset$. Note that the existence of D^0 is guaranteed by Assumption 4.1 and by Proposition 3.4. There are different methods to find an initial dictionary and these will be discussed in Sect. 4.8. For now, assume that D^0 is given. By Proposition 3.4, the basic solution x^0 corresponding to D^0 is a maximizer to (P). As part of the initialization, we find an index set of entering variables J^{D^0} as defined by (5).

Throughout the algorithm, for each dictionary D with given basic variables $\mathcal{B}$, optimality region $\Lambda^D \cap \Lambda$, and index set of entering variables J^D, we select an entering variable x_j, $j \in J^D$, and pick analog to the standard simplex method a leaving variable x_i satisfying

$$i \in \underset{\substack{i \in \mathcal{B} \\ (B^{-1}N)_{ij} > 0}}{\arg\min} \frac{(B^{-1}b)_i}{(B^{-1}N)_{ij}}, \tag{6}$$

whenever there exists some i with $(B^{-1}N)_{ij} > 0$. Here, indices i, j are written on behalf of the entries that correspond to the basic variable x_i and the nonbasic variable x_j, respectively. Note that this rule of picking leaving variables, together with the initialization of the algorithm with a primal feasible dictionary D^0, guarantees that each dictionary throughout the algorithm is primal feasible.

If there exists a basic variable x_i with $(B^{-1}N)_{ij} > 0$ satisfying (6), we perform the pivot $x_j \leftrightarrow x_i$ to form the dictionary $\bar{D}$ with basic variables $\bar{\mathcal{B}} = (\mathcal{B} \cup \{j\}) \setminus \{i\}$ and nonbasic variables $\bar{\mathcal{N}} = (\mathcal{N} \cup \{i\}) \setminus \{j\}$. For dictionary $\bar{D}$, we have $I_i^{\bar{D}} = \operatorname{cl}(I_j^D)^C = \{\lambda \in \mathbb{R}^{q-1} |\ w(\lambda)^T Z_{\bar{\mathcal{N}}}^T e^j \leq 0\}$. If dictionary $\bar{D}$ is considered at some point in the algorithm, it is known that the pivot $x_i \leftrightarrow x_j$ will yield the dictionary D considered above. Thus, we call (i, j) an *explored pivot (or direction)* for $\bar{D}$. We denote the set of all explored pivots of dictionary $\bar{D}$ by $E^{\bar{D}}$.

4.5 Detecting unbounded problems and constructing the set $\bar{\mathcal{X}}^h$

Now, consider the case where there is no candidate leaving variable for an entering variable x_j, $j \in J^D$, of dictionary D, that is, $(B^{-1}Ne^j) \leq 0$. As one can not perform a pivot, it is not possible to go beyond the halfspace I_j^D. Indeed, the parametrized problem (P_λ) is unbounded for $\lambda \notin I_j^D$. The following proposition shows that in that case, a direction of the recession cone of the lower image can be found from the current dictionary D, see Remark 3.3.

Proposition 4.3 *Let D be a dictionary with basic and nonbasic variables $\mathcal{B}$ and $\mathcal{N}$, $\Lambda^D \cap \Lambda$ be its optimality region satisfying $\Lambda^D \cap \operatorname{int} \Lambda \neq \emptyset$, and J^D be an index set of entering variables. If for some $j \in J^D$, $(B^{-1}Ne^j) \leq 0$, then the direction x^h defined by setting $x^h_{\mathcal{B}} = -B^{-1}Ne^j$ and $x^h_{\mathcal{N}} = e^j$ is a maximizer to (P) and $P^T x^h = -Z^T_{\mathcal{N}} e^j$.*

Proof Assume $(B^{-1}Ne^j) \leq 0$ for $j \in J^D$ and define x^h by setting $x^h_{\mathcal{B}} = -B^{-1}Ne^j$ and $x^h_{\mathcal{N}} = e^j$. By definition, the direction x^h would be a maximizer for (P) if and only if it is a (point) maximizer for the homogeneous problem (P^h), see Sect. 3. It holds

$$[A \;\; I]x^h = Bx^h_{\mathcal{B}} + Nx^h_{\mathcal{N}} = 0.$$

Moreover, $x^h_{\mathcal{N}} = e^j \geq 0$ and $x^h_{\mathcal{B}} = -B^{-1}Ne^j \geq 0$ by assumption. Thus, x^h is primal feasible for problem (P^h) and also for problem ($\mathrm{P}^h_1(w(\lambda))$) for all $\lambda \in \Lambda$, that is, $x^h \in \mathcal{X}^h \backslash \{0\}$. Let $\lambda \in \Lambda^D \cap \operatorname{int} \Lambda$, which implies $w(\lambda) \in \operatorname{ri} W \subseteq \operatorname{int} C^+$. Note that by definition of the optimality region, it is true that $w(\lambda)^T Z^T_{\mathcal{N}} \geq 0$. Thus, x^h is also dual feasible for ($\mathrm{P}^h_1(w(\lambda))$) and it is an optimal solution for the parametrized homogeneous problem for $\lambda \in \Lambda^D \cap \operatorname{int} \Lambda$. By Proposition 3.4 (applied to (P^h) and ($\mathrm{P}^h_1(w(\lambda))$)), x^h is a maximizer of (P^h). The value of the objective function of (P^h) at x^h is given by

$$[P^T \;\; 0]x^h = P^T_{\mathcal{B}} x^h_{\mathcal{B}} + P^T_{\mathcal{N}} x^h_{\mathcal{N}} = -Z^T_{\mathcal{N}} e^j.$$

□

Remark 4.4 If for an entering variable x_j, $j \in J^D$, of dictionary D, there is no candidate leaving variable, we conclude that problem (P) is unbounded in the sense of Definition 3.2. Then, in addition to the set of point maximizers $\bar{\mathcal{X}}$ one also needs to find the set of (direction) maximizers $\bar{\mathcal{X}}^h$ of (P), which by Proposition 4.3 can be obtained by collecting directions x^h defined by $x^h_{\mathcal{B}} = -B^{-1}Ne^j$ and $x^h_{\mathcal{N}} = e^j$ for every $j \in J^D$ with $B^{-1}Ne^j \leq 0$ for all dictionaries visited throughout the algorithm. For an index set J^D of entering variables of each dictionary D, we denote the set of indices of entering variables with no candidate leaving variable for dictionary D by $J^D_b := \{j \in J^D |\; B^{-1}Ne^j \leq 0\}$. In other words, $J^D_b \subseteq J^D$ is such that for any $j \in J^D_b$, (P_λ) is unbounded for $\lambda \notin I^D_j$.

4.6 Partition of Λ: putting it all together

We have seen in the last subsections that basic solutions of dictionaries visited by the algorithm yield (weak) point maximizers of (P) and partition Λ into optimality regions for bounded problems (P_λ), while encountering an entering variable with no leaving variable in a dictionary yields direction maximizers of (P) as well as regions of Λ corresponding to unbounded problems (P_λ). This will be the basic idea to construct the two sets $\bar{\mathcal{X}}$ and $\bar{\mathcal{X}}^h$ and to obtain a partition of the parameter set Λ. In order to show that $(\bar{\mathcal{X}}, \bar{\mathcal{X}}^h)$ produces a solution to (P), one still needs to ensure finiteness of the procedure, that the whole set Λ is covered, and that the basic solutions of dictionaries visited yield not only weak point maximizers of (P), but point maximizers.

Observe that whenever x_j, $j \in J^D$, is the entering variable for dictionary D with $\Lambda^D \cap \Lambda \neq \emptyset$ and there exists a leaving variable x_i, the optimality region $\Lambda^{\bar{D}}$ for dictionary $\bar{D}$ after the pivot is guaranteed to be non-empty. Indeed, it is easy to show that

$$\emptyset \subsetneq \Lambda^{\bar{D}} \cap \Lambda^D \subseteq \{\lambda \in \mathbb{R}^{q-1} \mid w(\lambda)^T Z_{\mathcal{N}} e^j = 0\},$$

where $\mathcal{N}$ is the collection of nonbasic variables of dictionary D. Moreover, the basic solutions read from dictionaries D and $\bar{D}$ are both optimal solutions to the parametrized problem (P_λ) for $\lambda \in \Lambda^{\bar{D}} \cap \Lambda^D$. Note that the common optimality region of the two dictionaries has no interior.

Remark 4.5 a. In some cases it is possible that $\Lambda^{\bar{D}}$ itself has no interior and it is a subset of the neighboring optimality regions corresponding to some other dictionaries.

b. Even though it is possible to come across dictionaries with optimality regions having empty interior, for any dictionary D found during the algorithm $\Lambda^D \cap \operatorname{int} \Lambda \neq \emptyset$ holds. This is guaranteed by starting with a dictionary D^0 satisfying $\Lambda^{D^0} \cap \operatorname{int} \Lambda \neq \emptyset$ together with the rule of selecting the entering variables, see (5). More specifically, throughout the algorithm, whenever I_j^D corresponds to the boundary of Λ it is guaranteed that $j \notin J^D$. By this observation and by Proposition 3.4, it is clear that the basic solution corresponding to the dictionary D is not only a weak maximizer but it is a maximizer.

Let us denote the set of all primal feasible dictionaries D satisfying $\Lambda^D \cap \operatorname{int} \Lambda \neq \emptyset$ by $\mathcal{D}$. Note that $\mathcal{D}$ is a finite collection. Let the set of parameters $\lambda \in \Lambda$ yielding bounded scalar problems (P_λ) be Λ_b. Then it can easily be shown that

$$\begin{aligned} \Lambda_b &:= \{\lambda \in \Lambda \mid (\mathrm{P}_\lambda) \text{ has an optimal solution}\} \\ &= \bigcup_{D \in \mathcal{D}} (\Lambda^D \cap \Lambda). \end{aligned} \tag{7}$$

Note that not all dictionaries in $\mathcal{D}$ are required to be known in order to cover Λ_b. First, the dictionaries mentioned in Remark 4.5 a. do not provide a new region within Λ_b. One should keep in mind that the algorithm may still need to visit some of these dictionaries in order to go beyond the optimality region of the current one. Secondly, in case there are multiple possible leaving variables for the same entering variable, instead of performing all possible pivots, it is enough to pick one leaving variable and continue with this choice. Indeed, choosing different leaving variables leads to different partitions of the same subregion within Λ_b.

By this observation, it is clear that there is a subcollection of dictionaries $\bar{\mathcal{D}} \subseteq \mathcal{D}$ which defines a partition of Λ_b in the following sense

$$\bigcup_{D \in \bar{\mathcal{D}}} (\Lambda^D \cap \Lambda) = \Lambda_b. \tag{8}$$

If there is at least one dictionary $D \in \bar{\mathcal{D}}$ with $J_b^D \neq \emptyset$, it is known by Remark 4.4 that (P) is unbounded. If further Λ_b is connected, one can show that

$$\bigcap_{D \in \bar{\mathcal{D}},\ j \in J_b^D} (I_j^D \cap \Lambda) = \Lambda_b, \tag{9}$$

holds. Indeed, connectedness of Λ_b is correct, see Remark 4.7 below.

4.7 The algorithm

The aim of the parametrized simplex algorithm is to visit a set of dictionaries $\bar{\mathcal{D}}$ satisfying (8).

In order to explain the algorithm we introduce the following definition.

Definition 4.6 $D \in \mathcal{D}$ is said to be a *boundary dictionary* if Λ^D and an index set of entering variables J^D is known. A boundary dictionary is said to be *visited* if the resulting dictionaries of all possible pivots from D are boundary and the index set J_b^D corresponding to J^D (see Remark 4.4) is known.

The motivation behind this definition is to treat the dictionaries as nodes and the possible pivots between dictionaries as the edges of a graph. Note that more than one dictionary may correspond to the same maximizer.

Remark 4.7 The graph described above is not necessarily connected. However, there exists a connected subgraph which includes at least one dictionary corresponding to each maximizer found by visiting the whole graph. The proof for the case $C = \mathbb{R}^q_+$ is given in [28] and it can be generalized easily to any polyhedral ordering cone. Note that this implies that the set Λ_b is connected.

The idea behind the algorithm is to visit a sufficient subset of 'nodes' to cover the set Λ_b. This can be seen as a special online traveling salesman problem. Indeed, we employ the terminology used in [18]. The set of all 'currently' boundary and visited dictionaries through the algorithm are denoted by *BD* and *VS*, respectively.

The algorithm starts with $BD = \{D^0\}$ and $VS = \emptyset$, where D^0 is the initial dictionary with index set of entering variables J^{D^0}. We initialize $\bar{\mathcal{X}}^h$ as the empty set and $\bar{\mathcal{X}}$ as $\{x^0\}$, where x^0 is the basic solution corresponding to D^0. Also, as there are no explored directions for D^0 we set $E^{D^0} = \emptyset$.

For a boundary dictionary D, we consider each $j \in J^D$ and check the leaving variable corresponding to x_j. If there is no leaving variable, we add x^h defined by $x^h_{\mathcal{B}} = -B^{-1}Ne^j$, and $x^h_{\mathcal{N}} = e^j$ to the set $\bar{\mathcal{X}}^h$, see Proposition 4.3. Otherwise, a corresponding leaving variable x_i is found. If $(j, i) \notin E^D$, we perform the pivot $x_j \leftrightarrow x_i$ as it has not been explored before. We check if the resulting dictionary $\bar{D}$ is marked as visited or boundary. If $\bar{D} \in VS$, there is no need to consider $\bar{D}$ further. If $\bar{D} \in BD$, then (i, j) is added to the set of explored directions for $\bar{D}$. In both cases, we continue by checking some other entering variable of D. If $\bar{D}$ is neither visited nor boundary, then the corresponding basic solution $\bar{x}$ is added to the set $\bar{\mathcal{X}}$, an index set of entering variables $J^{\bar{D}}$ is computed, (i, j) is added to the set of explored directions $E^{\bar{D}}$, and $\bar{D}$ itself is added to the set of boundary dictionaries. Whenever all $j \in J^D$ have been considered, D becomes visited. Thus, D is deleted from the set BD and added to the set VS. The algorithm stops when there are no more boundary dictionaries.

Algorithm 1 Parametric Simplex Algorithm for LVOP

```
1: Find $D^0$ and an index set of entering variables $J^{D^0}$;
2: Initialize $\begin{cases} BD = \{D^0\}, \bar{\mathcal{X}} = \{x^0\}; \\ VS, \bar{\mathcal{X}}^h, E^{D^0} = \emptyset; \end{cases}$
3: while $BD \neq \emptyset$ do
4:   Let $D \in BD$ with nonbasic variables $\mathcal{N}$ and index set of entering variables $J^D$;
5:   for $j \in J^D$ do
6:     Let $x_j$ be the entering variable;
7:     if $B^{-1}Ne^j \leq 0$ then
8:       Let $x^h$ be such that $x^h_{\mathcal{B}} = -B^{-1}Ne^j$ and $x^h_{\mathcal{N}} = e^j$;
9:       $\bar{\mathcal{X}}^h \leftarrow \bar{\mathcal{X}}^h \cup \{x^h\}$;
10:      $P^T[\bar{\mathcal{X}}^h] \leftarrow P^T[\bar{\mathcal{X}}^h] \cup \{-Z^T_{\mathcal{N}} e^j\}$
11:    else
12:      Pick $i \in \arg\min_{i \in \mathcal{B},\ (B^{-1}N)_{ij} > 0} \frac{(B^{-1}b)_i}{(B^{-1}N)_{ij}}$;
13:      if $(j, i) \notin E^D$ then
14:        Perform the pivot with entering variable $x_j$ and leaving variable $x_i$;
15:        Call the new dictionary $\bar{D}$ with nonbasic variables $\bar{\mathcal{N}} = \mathcal{N} \cup \{i\} \backslash \{j\}$;
16:        if $\bar{D} \notin VS$ then
17:          if $\bar{D} \in BD$ then
18:            $E^{\bar{D}} \leftarrow E^{\bar{D}} \cup \{(i, j)\}$;
19:          else
20:            Let $\bar{x}$ be the basic solution for $\bar{D}$;
21:            $\bar{\mathcal{X}} \leftarrow \bar{\mathcal{X}} \cup \{\bar{x}\}$;
22:            $P^T[\bar{\mathcal{X}}] \leftarrow P^T[\bar{\mathcal{X}}] \cup \{P^T\bar{x}\}$;
23:            Compute an index set of entering variables $J^{\bar{D}}$ of $\bar{D}$;
24:            Let $E^{\bar{D}} = \{(i, j)\}$;
25:            $BD \leftarrow BD \cup \{\bar{D}\}$;
26:          end if
27:        end if
28:      end if
29:    end if
30:  end for
31:  $VS \leftarrow VS \cup \{D\}$,   $BD \leftarrow BD \backslash \{D\}$;
32: end while
33: return $\begin{cases} (\bar{\mathcal{X}}, \bar{\mathcal{X}}^h) & : \text{A finite solution of } (P); \\ (P^T[\bar{\mathcal{X}}], P^T[\bar{\mathcal{X}}^h] \cup \{y^1, \ldots, y^t\}) & : \text{V representation of } \mathcal{P}. \end{cases}$
```

Theorem 4.8 *Algorithm 1 returns a solution* $(\bar{\mathcal{X}}, \bar{\mathcal{X}}^h)$ *to (P).*

Proof Algorithm 1 terminates in a finite number of iterations since the overall number of dictionaries is finite and there is no risk of cycling as the algorithm never performs 'already explored pivots', see line 13. $\bar{\mathcal{X}}$, $\bar{\mathcal{X}}^h$ are finite sets of feasible points and directions, respectively, for (P), and they consist of only maximizers by Propositions 3.4 and 4.3 together with Remark 4.5 b. Hence, it is enough to show that $(\bar{\mathcal{X}}, \bar{\mathcal{X}}^h)$ satisfies (2).

Observe that by construction, the set of all visited dictionaries $\bar{\mathcal{D}} := VS$ at termination satisfies (8). Indeed, there are finitely many dictionaries and Λ_b is a connected set, see Remark 4.7. It is guaranteed by (8) that for any $w \in C^+$, for which $(\mathrm{P}_1(w))$ is bounded, there exists an optimal solution $\bar{x} \in \bar{\mathcal{X}}$ of $(\mathrm{P}_1(w))$. Then, it is clear that $(\bar{\mathcal{X}}, \bar{\mathcal{X}}^h)$ satisfies (2) as long as $R := \operatorname{cone} P^T[\bar{\mathcal{X}}^h] - C$ is the recession cone $\mathcal{P}_\infty$ of the lower image.

If for all $D \in \bar{\mathcal{D}}$ the set $J_b^D = \emptyset$, then (P) is bounded, $\bar{\mathcal{X}}^h = \emptyset$, and trivially $R = -C = \mathcal{P}_\infty$. For the general case, we show that $-\mathcal{P}_\infty^+ = -R^+$ which implies $\mathcal{P}_\infty = \operatorname{cone} P^T[\bar{\mathcal{X}}^h] - C$. Assume there is at least one dictionary $D \in \bar{\mathcal{D}}$ with $J_b^D \neq \emptyset$. Then, by Remarks 4.4 and 4.7, (P) is unbounded, $\bar{\mathcal{X}}^h \neq \emptyset$ and $\bar{\mathcal{D}}$ also satisfies (9). On the one hand, by definition of I_j^D, we can write (9) as

$$\Lambda_b = \bigcap_{D \in VS,\ j \in J_b^D} \left\{ \lambda \in \Lambda \mid w(\lambda)^T Z_{\mathcal{N}_D}^T e^j \geq 0 \right\}, \tag{10}$$

where $\mathcal{N}_D$ is the set of nonbasic variables corresponding to dictionary D. On the other hand, by construction and by Proposition 4.3, we have

$$R = \operatorname{cone}\left(\left\{ -Z_{\mathcal{N}_D}^T e^j \mid j \in \bigcup_{D \in VS} J_b^D \right\} \cup \left\{ -y^1, \ldots, -y^t \right\} \right),$$

where $\{y^1, \ldots, y^t\}$ is the set of generating vectors for the ordering cone C. The dual cone can be written as

$$R^+ = \bigcap_{D \in VS,\ j \in J_b^D} \left\{ w \in \mathbb{R}^q \mid w^T Z_{\mathcal{N}_D}^T e^j \leq 0 \right\} \cap \bigcap_{i=1}^{k} \left\{ w \in \mathbb{R}^q \mid w^T y^i \leq 0 \right\}. \tag{11}$$

Now, let $w \in -\mathcal{P}_\infty^+$. By proposition 3.5, $(\mathrm{P}_1(w))$ has an optimal solution. As $c^T w > 0$, also $\left(P_1\left(\frac{w}{c^T w}\right)\right) = (P_{\bar{\lambda}})$ has an optimal solution, where $\bar{\lambda} := \frac{1}{c^T w}(w_1, \ldots, w_{q-1})^T$ and thus $w(\bar{\lambda}) = \frac{w}{c^T w}$. By the definition of Λ_b given by (7), $\bar{\lambda} \in \Lambda_b$. Then, by (10), $\bar{\lambda} \in \Lambda$ and $w(\lambda)^T Z_{\mathcal{N}_D}^T e^j \geq 0$ for all $j \in J_b^D, D \in VS$. This holds if and only if $w \in -R_+$ by definition of Λ and by (11). The other inclusion can be shown symmetrically. □

Remark 4.9 In general, simplex-type algorithms are known to work better if the problem is not degenerate. If the problem is degenerate, Algorithm 1 may find redundant maximizers. The effects of degeneracy will be provided in more detail in Sect. 7. For now, let us mention that it is possible to eliminate the redundancies by additional steps in Algorithm 1. There are two types of redundant maximizers.

a. Algorithm 1 may find multiple point (direction) maximizers that are mapped to the same point (direction) in the image space. In order to find a solution that is free of these type of redundant maximizers, one may change line 21 (9) of the algorithm such that the current maximizer x (x^h) is added to the set $\bar{\mathcal{X}}$ ($\bar{\mathcal{X}}^h$) only if its image is not in the current set $P^T[\bar{\mathcal{X}}]$ ($P^T[\bar{\mathcal{X}}^h]$).
b. Algorithm 1 may also find maximizers whose image is not a vertex on the lower image. One can eliminate these maximizers from the set $\bar{\mathcal{X}}$ by performing a vertex elimination at the end.

4.8 Initialization

There are different ways to initialize Algorithm 1. We provide two methods, both of which also determine if the problem has no solution. Note that (P) has no solution if $\mathcal{X} = \emptyset$ or if the lower image is equal to $\mathbb{R}^q$, that is, if ($\text{P}_1(w)$) is unbounded for all $w \in \operatorname{int} C^+$. We assume $\mathcal{X}$ is nonempty. Moreover, for the purpose of this section, we assume without loss of generality that $b \geq 0$. Indeed, if $b \ngeq 0$, one can find a primal feasible dictionary by applying any 'Phase 1' algorithm that is available for the usual simplex method, see [32].

The first initialization method finds a weight vector $w^0 \in \operatorname{int} C^+$ such that ($\text{P}_1(w^0)$) has an optimal solution. Then the optimal dictionary for ($\text{P}_1(w^0)$) is used to construct the initial dictionary D^0. There are different ways to choose the weight vector w^0. The second method of initialization can be thought of as a Phase 1 algorithm. It finds an initial dictionary as long as there exists one.

4.8.1 Finding w^0 and constructing D^0

The first way to initialize the algorithm requires finding some $w^0 \in \operatorname{int} C^+$ such that ($\text{P}_1(w^0)$) has an optimal solution. It is clear that if the problem is known to be bounded, then any $w^0 \in \operatorname{int} C^+$ works. However, it is a nontrivial procedure in general. In the following we give two different methods to find such w^0. The first method can also determine if the problem has no solution.

a. The first approach is to extend the idea presented in [15] to any solid polyhedral pointed ordering cone C. Accordingly, finding w^0 involves solving the following linear program:

$$\begin{aligned} \text{minimize} \quad & b^T u && (\text{P}_0) \\ \text{subject to} \quad & A^T u - P w \geq 0, \\ & Y^T (w - c) \geq 0, \\ & u \geq 0, \end{aligned}$$

where $c \in \operatorname{int} C^+$, and the columns of Y are the generating vectors of C. Under the assumption $b \geq 0$, it is easy to show that (P) has a maximizer if and only if (P_0) has an optimal solution. Note that (P_0) is bounded. If (P_0) is infeasible, then we conclude that the lower image has no vertex and (P) has no solution. In case it has an optimal solution (u^*, w^*), then one can take $w^0 = \frac{w^*}{c^T w^*} \in \operatorname{int} C^+$, and solve ($\mathrm{P}_1(w^0)$) optimally. For the randomly generated examples of Sect. 5 we have used this method.

b. Using the idea provided in [27], it might be possible to initialize the algorithm without even solving a linear program. By the structure of a particular problem, one may start with a dictionary which is trivially optimal for some weight w^0. In this case, one can start with this choice of w^0, and get the initial dictionary D^0 even without solving an LP. An example is provided in Sect. 5, see Remark 5.3.

In order to initialize the algorithm, w^0 can be used to construct the initial dictionary D^0. Without loss of generality assume that $c^T w^0 = 1$, indeed one can always normalize since $c \in \operatorname{int} C$ implies $c^T w^0 > 0$. Then, clearly $w^0 \in \operatorname{ri} W$. Let $\mathcal{B}^0$ and $\mathcal{N}^0$ be the set of basic and nonbasic variables corresponding to the optimal dictionary D^* of ($\mathrm{P}_1(w^0)$). If one considers the dictionary D^0 for (P_λ) with the basic variables $\mathcal{B}^0$ and nonbasic variables $\mathcal{N}^0$, the objective function of D^0 will be different from D^* as it depends on the parameter λ. However, the matrices B^0, N^0, and hence the corresponding basic solution x^0, are the same in both dictionaries. We consider D^0 as the initial dictionary for the parametrized problem (P_λ). Note that B^0 is a nonsingular matrix as it corresponds to dictionary D^*. Moreover, since D^* is an optimal dictionary for ($\mathrm{P}_1(w^0)$), x^0 is clearly primal feasible for (P_λ) for any $\lambda \in \mathbb{R}^{q-1}$. Furthermore, the optimality region of D^0 satisfies $\Lambda^{D^0} \cap \operatorname{int} \Lambda \neq \emptyset$ as $\lambda^0 := [w_1^0, \dots, w_{q-1}^0] \in \Lambda^{D^0} \cap \operatorname{int} \Lambda$. Thus, x^0 is also dual feasible for (P_λ) for $\lambda \in \Lambda^{D^0}$, and x^0 is a maximizer to (P).

4.8.2 Perturbation method

The second method of initialization works similar to the idea presented for Algorithm 1 itself.

Assuming that $b \geq 0$, problem (P_λ) is perturbed by an additional parameter $\mu \in \mathbb{R}$ as follows:

$$\begin{aligned} \text{maximize} \quad & (w(\lambda)^T P^T - \mu \mathbf{1}^T)x \\ \text{subject to} \quad & Ax \leq b, \\ & x \geq 0, \end{aligned} \tag{$\mathrm{P}_{\lambda,\mu}$}$$

where $\mathbf{1}$ is the vector of ones. After introducing the slack variables, consider the dictionary with basic variables $x_{n+1}, \dots, x_{m+n}$ and with nonbasic variables $x_1, \dots, x_n$. This dictionary is primal feasible as $b \geq 0$. Moreover, it is dual feasible if $Pw(\lambda) - \mu \mathbf{1} \leq 0$. We introduce the optimality region of this dictionary as

$$M^0 := \{(\lambda, \mu) \in \Lambda \times \mathbb{R}_+ |\ Pw(\lambda) - \mu \mathbf{1} \leq 0\}.$$

Note that M^0 is not empty as μ can take sufficiently large values.

The aim of the perturbation method is to find an optimality region M such that

$$M \cap \operatorname{ri}(\Lambda \times \{0\}) \neq \emptyset, \tag{12}$$

where $\Lambda \times \{0\} := \{(\lambda, 0) |\ \lambda \in \Lambda\}$. If the current dictionary satisfies (12), then it can be taken as an initial dictionary D^0 for Algorithm 1 after deleting the parameter μ. Otherwise, the defining inequalities of the optimality region are found. Clearly, they correspond to the entering variables of the current dictionary. The search for an initial dictionary continues similar to the original algorithm. Note that if there does not exist a leaving variable for an entering variable, ($P_{\lambda,\mu}$) is found to be unbounded for some set of parameters. The algorithm continues until we obtain a dictionary for which the optimality region satisfies (12) or until we cover the the parameter set $\Lambda \times \mathbb{R}_+$ by the optimality regions and by the regions that are known to yield unbounded problems. At termination, if there exist no dictionary that satisfies (12), then we conclude that there is no solution to problem (P). Otherwise, we initialize the algorithm with D^0. See Example 5.1, Remark 5.4.

5 Illustrative examples

We provide some examples and numerical results in this section. The first example illustrates how the different methods of initialization and the algorithm work. The second example shows that Algorithm 1 can find a solution even though the lower image does not have any vertices.

Example 5.1 Consider the following problem

$$\begin{aligned} \text{maximize} \quad & (x_1, x_2 - x_3, x_3)^T \text{ with respect to } \leq_{\mathbb{R}^3_+} \\ \text{subject to} \quad & x_1 + x_2 \leq 5 \\ & x_1 + 2x_2 - x_3 \leq 9 \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

Let $c = (1, 1, 1)^T \in \operatorname{int} \mathbb{R}^3_+$. Clearly, we have $\Lambda = \{\lambda \in \mathbb{R}^2 |\ \lambda_1 + \lambda_2 \leq 1,\ \lambda_i \geq 0,\ \ i = 1, 2\}$.

Let us illustrate the different initialization methods.

Remark 5.2 (Initializing by solving ($\mathbf{P_0}$), see Sect. 4.8.1 a.) A solution of (P_0) is found as $w^* = (1, 1, 1)^T$. Then, we take $w^0 = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})^T$ as the initial weight vector. $x^0 = (5, 0, 0)^T$ is an optimal solution found for $P_1(w^0)$. The indices of the basic variables of the corresponding optimal dictionary are $\mathcal{B}^0 = \{1, 5\}$. We form dictionary D^0 of problem (P_λ) with basic variables $\mathcal{B}^0$:

$$\begin{array}{rclllll} \xi & = & 5\lambda_1 & -\lambda_1 x_4 & -(\lambda_1 - \lambda_2)x_2 & -(\lambda_1 + 2\lambda_2 - 1)x_3 \\ x_1 & = & 5 & -x_4 & -x_2 & \\ x_5 & = & 4 & +x_4 & -x_2 & +x_3 \end{array}$$

Remark 5.3 (Initializing using the structure of the problem, see Sect. 4.8.1b.) The structure of Example 5.1 allows us to initialize without solving a linear program. Consider $w^0 = (1, 0, 0)^T$. As the objective of $\mathrm{P}_1(w^0)$ is to maximize x_1 and the most restraining constraint is $x_1 + x_2 \le 5$ together with $x_i \ge 0$, $x = (5, 0, 0)^T$ is an optimal solution of $\mathrm{P}_1(w^0)$. The corresponding slack variables are $x_4 = 0$ and $x_5 = 4$. Note that this corresponds to the dictionary with basic variables $\{1, 5\}$ and nonbasic variables $\{2, 3, 4\}$, which yields the same initial dictionary D^0 as above. Note that one needs to be careful as $w^0 \notin \operatorname{int} C^+$ but $w^0 \in \operatorname{bd} C^+$. In order to ensure that the corresponding initial solution is a maximizer and not only a weak maximizer, one needs to check the optimality region of the initial dictionary. If the optimality region has a nonempty intersection with $\operatorname{int} \Lambda$, which is the case for D^0, then the corresponding basic solution is a maximizer. In general, if one can find $w \in \operatorname{int} C^+$ such that $(\mathrm{P}_1(w))$ has a trivial optimal solution, then the last step is clearly unnecessary.

Remark 5.4 (Initializing using the perturbation method, see Sect. 4.8.2.) The starting dictionary for $(\mathrm{P}_{\lambda,\mu})$ of the perturbation method is given as

$$\begin{array}{llllll}
\xi & = & -(\mu-\lambda_1)x_1 & -(\mu-\lambda_2)x_2 & -(\mu+\lambda_1+2\lambda_2-1)x_3 \\
x_4 & = 5 & -x_1 & -x_2 & \\
x_5 & = 9 & -x_1 & -2x_2 & +x_3
\end{array}$$

This dictionary is optimal for $M^0 = \{(\lambda, \mu) \in \Lambda \times \mathbb{R} |\ \mu - \lambda_1 \ge 0,\ \mu - \lambda_2 \ge 0,\ \mu + \lambda_1 + 2\lambda_2 \ge 1\}$. Clearly, M^0 does not satisfy (12). The defining halfspaces for M^0 correspond to the nonbasic variables x_1, x_2 and x_3. If x_1 enters, then the leaving variable is x_4 and the next dictionary has the optimality region $M^1 = \{(\lambda, \mu) \in \Lambda \times \mathbb{R} |\ -\mu + \lambda_1 \ge 0,\ \lambda_1 - \lambda_2 \ge 0,\ \mu + \lambda_1 + 2\lambda_2 \ge 1\}$ which satisfies (12). Then, by deleting μ the initial dictionary is found to be D^0 as above. Different choices of entering variables in the first iteration might yield different initial dictionaries.

Consider the initial dictionary D^0. Clearly, $I_4^{D^0} = \{\lambda \in \mathbb{R}^2 |\ \lambda_1 \ge 0\}$, $I_2^{D^0} = \{\lambda \in \mathbb{R}^2 |\ \lambda_1 - \lambda_2 \ge 0\}$, and $I_3^{D^0} = \{\lambda \in \mathbb{R}^2 |\ \lambda_1 + 2\lambda_2 \ge 1\}$. The defining halfspaces for $\Lambda^{D^0} \cap \Lambda$ correspond to the nonbasic variables x_2, x_3, thus we have $J^{D^0} = \{2, 3\}$.

The iteration starts with the only boundary dictionary D^0. If x_2 is the entering variable, x_5 is picked as the leaving variable. The next dictionary, D^1, has basic variables $\mathcal{B}^1 = \{1, 2\}$, the basic solution $x^1 = (1, 4, 0)^T$, and a parameter region $\Lambda^{D^1} = \{\lambda \in \mathbb{R}^2 |\ 2\lambda_1 - \lambda_2 \ge 0,\ -\lambda_1 + \lambda_2 \ge 0,\ 2\lambda_1 + \lambda_2 \ge 1\}$. The halfspaces corresponding to the nonbasic variables x_j, $j \in J^{D^1} = \{3, 4, 5\}$ are defining for the optimality region $\Lambda^{D^1} \cap \Lambda$. Moreover, $E^{D^1} = \{(5, 2)\}$ is an explored pivot for D^1.

From dictionary D^0, for entering variable x_3 there is no leaving variable according to the minimum ratio rule (6). We conclude that problem (P_λ) is unbounded for $\lambda \in \mathbb{R}^2$ such that $\lambda_1 + 2\lambda_2 < 1$. Note that

$$B^{-1}N = \begin{bmatrix} 1 & 1 & 0 \\ -1 & 1 & -1 \end{bmatrix},$$

and the third column corresponds to the entering variable x_3. Thus, $x^h_{\mathcal{B}^0} = (x^h_1, x^h_5)^T = (0, -1)^T$ and $x^h_{\mathcal{N}^0} = (x^h_4, x^h_2, x^h_3)^T = e_3 = (0, 0, 1)^T$. Thus, we add $x^h = (x^h_1, x^h_2, x^h_3) = (0, 0, 1)^T$ to the set $\bar{\mathcal{X}}^h$, see Algorithm 1, line 12. Also, by Proposition 4.3, $P^T x^h = (0, -1, 1)^T$ is an extreme direction of the lower image $\mathcal{P}$. After the first iteration, we have $VS = \{D^0\}$, and $BD = \{D^1\}$.

For the second iteration, we consider $D^1 \in BD$. There are three possible pivots with entering variables x_j, $j \in J^{D^1} = \{3, 4, 5\}$. For x_5, x_2 is found as the leaving variable. As $(5, 2) \in E^{D^1}$, the pivot is already explored and not necessary. For x_3, the leaving variable is found as x_1. The resulting dictionary D^2 has basic variables $\mathcal{B}^2 = \{2, 3\}$, basic solution $x^2 = (0, 5, 1)^T$, the optimality region $\Lambda^{D^2} \cap \Lambda = \{\lambda \in \mathbb{R}^2_+ |\ 2\lambda_1 + \lambda_2 \leq 1,\ 2\lambda_1 + 3\lambda_2 \leq 2,\ -\lambda_1 - 2\lambda_2 \leq -1\}$, and the indices of the entering variables $J^{D^2} = \{1, 4, 5\}$. Moreover, we write $E^{D^2} = \{(1, 3)\}$.

We continue the second iteration by checking the entering variable x_4 from D^1. The leaving variable is found as x_1. This pivot yields a new dictionary D^3 with basic variables $\mathcal{B}^3 = \{2, 4\}$ and basic solution $x^3 = (0, 4.5, 0)^T$. The indices of the entering variables are found as $J^{D^3} = \{1, 3\}$. Also, we have $E^{D^3} = \{(1, 4)\}$. At the end of the second iteration we have $VS = \{D^0, D^1\}$, and $BD = \{D^2, D^3\}$.

Consider $D^2 \in BD$ for the third iteration. For x_1, the leaving variable is x_3 and we obtain dictionary D^1 which is already visited. For x_4, the leaving variable is x_3. We obtain the boundary dictionary D^3 and update the explored pivots for it as $E^{D^3} = \{(1, 4), (3, 4)\}$. Finally, for entering variable x_5 there is no leaving variable and one finds the same x^h that is already found at the first iteration. At the end of third iteration, $VS = \{D^0, D^1, D^2\}$, $BD = \{D^3\}$.

For the next iteration, D^3 is considered. The pivots for both entering variables yield already explored ones. At the end of this iteration there are no more boundary dictionaries and the algorithm terminates with $VS = \{D^0, D^1, D^2, D^3\}$. Figure 1 shows the optimality regions after the four iterations. The color blue indicates that the corresponding dictionary is visited, yellow stands for boundary dictionaries and the gray region corresponds to the set of parameters for which problem (P$_\lambda$) is unbounded.

The solution to the problem is $(\bar{\mathcal{X}}, \bar{\mathcal{X}}^h)$ where $\bar{\mathcal{X}} = \{(5, 0, 0)^T, (1, 4, 0)^T, (0, 5, 1)^T, (0, 4.5, 0)^T\}$, and $\bar{\mathcal{X}}^h = \left\{(0, 0, 1)^T\right\}$. The lower image can be seen in Fig. 2.

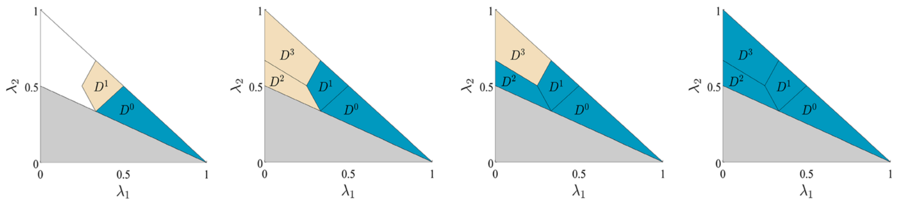

Fig. 1 Optimality regions after the first four iterations of Example 5.1

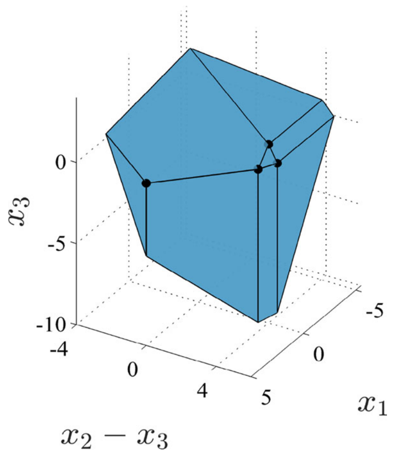

Fig. 2 Lower image $\mathcal{P}$ of Example 5.1

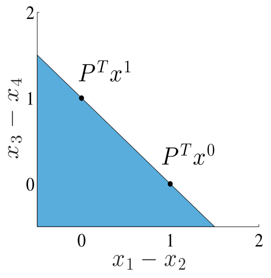

Fig. 3 Lower image $\mathcal{P}$ of Example 5.5

Example 5.5 Consider the following example.

$$\begin{aligned} &\text{maximize} \quad (x_1 - x_2, x_3 - x_4)^T \quad \text{with respect to} \quad \leq_{\mathbb{R}^2_+} \\ &\text{subject to} \quad x_1 - x_2 + x_3 - x_4 \leq 1 \\ &\qquad\qquad\qquad x_1, x_2, x_3, x_4 \geq 0. \end{aligned}$$

Let $c = (1, 1)^T \in \operatorname{int} \mathbb{R}^2_+$. Clearly, we have $\Lambda = [0, 1] \subseteq \mathbb{R}$. Using the method described in Sect. 4.8.1 a. we find $w^0 = (\frac{1}{2}, \frac{1}{2})$ as the initial scalarization parameter. Then, $x^0 = (1, 0, 0, 0)^T$ is an optimal solution to $\mathrm{P}_1(w^0)$ and the index set of the basic variables of D^0 is found as $\mathcal{B}^0 = \{1\}$. Algorithm 1 terminates after two iterations and yields $\bar{\mathcal{X}} = \{(1, 0, 0, 0)^T, (0, 0, 1, 0)^T\}$, $\bar{\mathcal{X}}^h = \{(1, 0, 0, 1)^T, (0, 1, 1, 0)^T\}$. The lower image can be seen in Fig. 3. Note that as it is possible to generate the lower image only with one point maximizer, the second one is redundant, see Remark 4.9 b.

6 Comparison of different simplex algorithms for LVOP

As briefly mentioned in Sect. 1, there are different simplex algorithms to solve LVOPs. Among them, the Evans–Steuer algorithm [15] works very similar to the algorithm provided here. It moves from one dictionary to another where each dictionary gives a point maximizer. Moreover, it finds 'unbounded efficient edges', which correspond to the direction maximizers. Even though the two algorithms work in a similar way, they have some differences that affect the efficiency of the algorithms significantly. The main difference is that the Evans–Steuer algorithm finds the set of all maximizers whereas Algorithm 1 finds only a subset of maximizers, which generates a solution in the sense of Löhne [19] and allows to generate the set of all maximal elements of the image of the feasible set. In general, the Evans–Steuer algorithm visits more dictionaries than Algorithm 1 especially if the problem is degenerate.

First of all, in each iteration of Algorithm 1, for each entering variable x_j, only one leaving variable is picked among the set of all possible leaving variables, see line 12. Differently, the Evans–Steuer algorithm performs pivots $x_j \leftrightarrow x_i$ for all possible leaving variables, $i \in \arg\min_{i\in\mathcal{B},\, (B^{-1}N)_{ij}>0} \frac{(B^{-1}b)_i}{(B^{-1}N)_{ij}}$. If the problem is degenerate, this procedure leads the Evans–Steuer algorithm to visit many more dictionaries than Algorithm 1 does. In general, these additionally visited dictionaries yield maximizers that are already found. In [1,2], it has been shown that using the lexicographic rule to choose the leaving variables would be sufficient to cover all the efficient basic solutions. For the numerical tests that we run, see Sect. 7, we have modified the Evans–Steuer algorithm such that it uses the lexicographic rule.

Another difference between the two simplex algorithms is at the step where the entering variables are selected. In Algorithm 1, the entering variables are the ones which correspond to the defining inequalities of the current optimality region. Different methods to find the entering variables are provided in Sect. 4.3. The method that is employed for the numerical tests of Sect. 7 involves solving sequential LP's with $q-1$ variables and at most $n+t$ inequality constraints, where t is the number of generating vectors of the ordering cone. Note that the number of constraints are decreasing in each LP as one solves them successively. For each dictionary, the total number of LPs to solve is at most n in each iteration. The Evans–Steuer algorithm finds a larger set of entering variables, namely 'efficient nonbasic variables' for each dictionary. In order to find this set, it solves n LPs with $n+q+1$ variables, q equality and $n+q+1$ non-negativity constraints. More specifically, for each nonbasic variable $j \in \mathcal{N}$ it solves

$$\begin{aligned} \text{maximize} \quad & \mathbf{1}^T v \\ \text{subject to} \quad & Z_{\mathcal{N}}^T y - \delta Z_{\mathcal{N}}^T e^j - v = 0, \\ & y, \delta, v \geq 0, \end{aligned}$$

where $y \in \mathbb{R}^n, \delta \in \mathbb{R}, v \in \mathbb{R}^q$. Only if this program has an optimal solution 0, then x_j is an efficient nonbasic variable. This procedure is clearly costlier than the one employed in Algorithm 1. In [17], this idea is improved so that it is possible to complete the procedure by solving fewer LPs of the same structure. Further improvements are done also in [9]. Moreover, in [1,2] a different method is applied in order to find the efficient nonbasic variables. Accordingly, one needs to solve n LPs with $2q$ variables, n equality and $2q$ nonnegativity constraints. Clearly, this method is more efficient than the one used for the Evans–Steuer algorithm. However, the general idea of finding the efficient nonbasic variables clearly yields visiting more redundant dictionaries than Algorithm 1 would visit. Some of these additionally visited dictionaries yield different maximizers that map into already found maximal elements in the objective space, see Example 6.1; while some of them yield non-vertex maximal elements in the objective space, see Example 6.2.

Example 6.1 Consider the following simple example taken from [28], in which it has been used to illustrate the Evans–Steuer algorithm.

$$\begin{aligned} \text{maximize} \quad & (3x_1 + x_2, 3x_1 - x_2)^T \text{ with respect to } \leq_{\mathbb{R}^2_+} \\ \text{subject to} \quad & x_1 + x_2 \leq 4 \\ & x_1 - x_2 \leq 4 \\ & x_3 \leq 4 \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

If one uses Algorithm 1, the solution is provided right after the initialization. The initial set of basic variables can be found as $\mathcal{B}^0 = \{1, 5, 6\}$, and the basic solution corresponding to the initial dictionary is $x^0 = (4, 0, 0)^T$. One can easily check that x^0 is optimal for all $\lambda \in \Lambda$. Thus, Algorithm 1 stops and returns the single maximizer. On the other hand, it is shown in [28] that the Evans–Steuer algorithm terminates only after performing another pivot to obtain a new maximizer $x^1 = (4, 0, 4)^T$. This is because, from the dictionary with basic variables $\mathcal{B}^0 = \{1, 5, 6\}$ it finds x_3 as an efficient nonbasic variable and performs one more pivot with entering variable x_3. Clearly the image of x^1 is again the same vertex $(4, 4)^T$ in the image space. Thus, in order to generate a solution in the sense of Definition 3.1, the last iteration is unnecessary.

Example 6.2 Consider the following example.

$$\begin{aligned} \text{maximize} \quad & (-x_1 - x_3, -x_2 - 2x_3)^T \text{ with respect to } \leq_{\mathbb{R}^2_+} \\ \text{subject to} \quad & -x_1 - x_2 - 3x_3 \leq -1 \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

First, we solve the example by Algorithm 1. Clearly, $\Lambda = [0, 1] \subseteq \mathbb{R}$. We find an initial dictionary D^0 with $\mathcal{B}^0 = \{1\}$, which yields the maximizer $x^0 = (1, 0, 0)^T$. One can easily see that index set of the defining inequalities of the optimality region can be chosen either as $J^{D^0} = \{2\}$ or $J^{D^0} = \{3\}$. Note that Algorithm 1 picks one of them and continues with it. In this example we get $J^{D^0} = \{2\}$, perform the pivot $x_2 \leftrightarrow x_1$ to get D^1 with $\mathcal{B}^1 = \{2\}$ and $x^1 = (0, 1, 0)^T$. From D^1, there are two choices of sets of entering variables and we set $J^{D^1} = \{1\}$. As the pivot $x_1 \leftrightarrow x_2$ is already explored, the algorithm terminates with $\bar{\mathcal{X}} = \{x^0, x^1\}$ and $\bar{\mathcal{X}}^h = \emptyset$.

When one solves the same problem by the Evans–Steuer algorithm, from D^0, both x_2 and x_3 are found as entering variables. When x_3 enters from D^0, one finds a new maximizer $x^2 = (0, 0, \frac{1}{3})^T$. Note that this yields a nonvertex maximal element on the lower image, see Fig. 4.

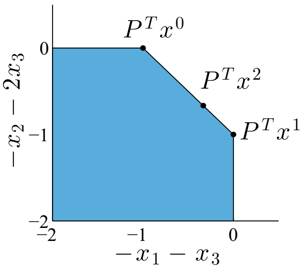

Fig. 4 Lower image $\mathcal{P}$ of Example 6.2

Remark 6.3 Note that if the problem is primal nondegenerate, then for a given entering variable of a given dictionary, both Algorithm 1 and the Evans–Steuer algorithm find the unique leaving variable. If in addition, every efficient nonbasic variable of a given dictionary corresponds to a defining inequality of its optimality region, then the entering variables from that dictionary would be the same for both algorithms. Indeed, the different type of redundancies that are explained in Remark 4.9 are mostly observed if there is a primal degeneracy or if there are efficient nonbasic variables which corresponds to redundant inequalities of the optimality region. Hence, it wouldn't be wrong to state that for 'nondegenerate' problems, the Evans–Steuer algorithm and Algorithm 1 follow similar paths. But for degenerate problems their performance will be quite different.

Apart from the Evans–Steuer algorithm Ehrgott, Puerto and Rodriguez-Chía [13] developed a primal–dual simplex algorithm to solve LVOPs. The algorithm finds a partition (Λ^d) of Λ. It is similar to Algorithm 1 in the sense that for each parameter set Λ^d, it provides an optimal solution x^d to the problems (P_λ) for all $\lambda \in \Lambda^d$. The difference between the two algorithms is in the method of finding the partition. The algorithm in [13] starts with a (coarse) partition of the set Λ. In each iteration it finds a finer partition until no more improvements can be done. In contrast to the algorithm proposed here, the algorithm in [13] requires solving in each iteration an LP with $n + m$ variables and l constraints where $m < l \leq m + n$, which clearly makes the algorithm computationally much more costly. In addition to solving one 'large' LP, it involves a procedure which is similar to finding the defining inequalities of a region given by a set of inequalities. Also, different from Algorithm 1, it finds only a set of weak maximizers so that as a last step one needs to perform a vertex enumeration in order to obtain a solution consisting of maximizers only. Finally, the algorithm provided in [13] can deal with unbounded problems only if the set Λ_b is provided, which requires a Phase 1 procedure.

7 Numerical results

In this section we provide numerical results to study the efficiency of Algorithm 1. We generate random problems, solve them with different algorithms and compare the solutions and the CPU times. Algorithm 1 is implemented in MATLAB. We also use a MATLAB implementation of Benson's algorithm, namely bensolve 1.2 [20]. The

Table 1 Run time statistics for randomly generated problems where $q = 3$. For the first row $n = 20$, $m = 40$; for the second row, $n = 30$, $m = 30$; for the last row $n = 40$, $m = 20$

min A	min B	min E	max A	max B	max E	avg A	avg B	avg E	#u
0.34	0.20	0.27	6.02	162.53	5.77	1.72	15.63	1.63	0
0.08	0.08	0.09	9.36	257.98	8.61	3.15	32.41	2.90	8
0.05	0.03	0.08	13.52	418.44	11.81	3.33	23.92	2.96	38

current version of bensolve 1.2 solves two linear programs in each iteration. However, we employ an improved version which solves only one linear program in each iteration, see [16,22,23]. For the Evans–Steuer algorithm, instead of using ADBASE [31], we implement the algorithm in MATLAB. This way, we can test the algorithms with the same machinery. This gives the opportunity to compare the CPU times. For each algorithm the linear programs are solved using the GLPK solver, see [25].

The first set of problems are randomly generated with no special structure. That is to say, these problems are not designed to be degenerate. In particular, each element of the matrices A and P and the vector b is sampled independently, the elements of A and P from a normal distribution with mean 0 and variance 100, and the elements of b from a uniform distribution over [0, 10]. As $b \geq 0$, we did not employ a Phase 1 algorithm to find a primal feasible initial dictionary. Table 1 shows the numerical results for the randomly generated problems with three objectives. We fix different numbers of variables (n) and constraints (m) and generate 100 problems for each size. We measure the average time that Algorithm 1 (avg A), bensolve 1.2. (avg B) and the Evans–Steuer algorithm (avg E) take to solve the problems. Moreover, we report the minimum (min A, min B, min E) and maximum (max A, max B, max E) running times for each algorithm among those 100 problems. The number of unbounded problems that are found among the 100 problems is denoted by #u.

Next, we randomly generate problems with four objectives and with different numbers of variables (n) and constraints (m). For each size we generate four problems. Table 2 shows the numerical results, where $|\bar{\mathcal{X}}|$ and $|\bar{\mathcal{X}}^h|$ are the number of elements of the set of point and direction maximizers, respectively. For each problem the time for Algorithm 1, for bensolve 1.2 and for the Evans–Steuer algorithm to terminate are shown by 'time A', 'time B' and 'time E', respectively.

For these particular examples all algorithms find the same solution $(\bar{\mathcal{X}}, \bar{\mathcal{X}}^h)$. As no structure is imposed on these problems, the probability that these problems are nondegenerate is very high. This explains finding the same solution by all of the algorithms. As seen from the Tables 1 and 2, the CPU times of the Evans–Steuer algorithm are very close to the CPU times of Algorithm 1 which is expected as explained in Remark 6.3.

As seen from Tables 1 and 2, the parametric simplex algorithm works more efficiently than bensolve 1.2 for the randomly generated problems. The main reason for the difference in the performances is that in each iteration, Benson's algorithm solves an LP that is in the same size of the original problem and also a vertex enumeration problem. Note that solving a vertex enumeration problem from scratch in each iteration is a costly procedure. In [6,12], an online vertex enumeration method has been proposed and this would increase the efficiency of Benson's algorithm.

Table 2 Computational results for randomly generated problems

q	n	m	$\lvert\bar{\mathcal{X}}\rvert$	$\lvert\bar{\mathcal{X}}^h\rvert$	Time A	Time B	Time E
4	30	50	267	0	3.91	64.31	3.61
4	30	50	437	0	6.95	263.39	7.06
4	30	50	877	0	15.73	1866.10	17.01
4	30	50	2450	0	74.98	33,507.00	73.69
4	40	40	814	0	20.41	1978.30	18.56
4	40	40	1468	81	42.39	11,785.00	38.20
4	40	40	2740	0	105.45	64,302.00	97.69
4	40	40	2871	324	121.16	82,142.00	112.11
4	50	30	399	21	10.53	233.11	9.23
4	50	30	424	0	11.22	294.17	9.92
4	50	30	920	224	28.08	3434.10	24.05
4	50	30	1603	176	55.97	14,550.00	49.86

Note that these randomly generated problems have no special structure and thus there is a high probability that these problems are nondegenerate. However, in general, Benson-type objective space algorithms are expected to be more efficient whenever the problem is degenerate. The main reason is that these algorithms do not need to deal with the different efficient solutions which map into the same point in the objective space. This, indeed, is one of the main motivation of Benson's algorithm for linear multiobjective optimization problems, see [5].

In order to see the efficiency of our algorithm for degenerate problems, we generate random problems which are designed to be degenerate. In the following examples this is done by generating a nonnegative b vector with many zero components and choosing objective functions with the potential to create optimality regions with empty interior within Λ. In particular, for the three-objective examples, we generate the first objective function randomly, take the second one to be the negative of the first objective function, and let the third objective consist of only one nonzero entry. For the four-objective examples, the first three objectives are created as described above and the fourth one is generated randomly in a way that at least half of its components are zero. The number of nonzero elements in b and in the last objective function of the four-objective problems are sampled independently from uniform distributions over the integers in the intervals $[0, \lfloor \frac{m}{2} \rfloor]$ and $[0, \lfloor \frac{q}{2} \rfloor]$, respectively. Each element of the first column and each possibly nonzero element of the third and the fourth column of P as well as each element of A and each possibly nonzero element of b is sampled independently, in the same way as for the nondegenerate problems.

First, we consider three objective functions where we fix different numbers of variables (n) and constraints (m). We generate 20 problems for each size. We measure the average time that Algorithm 1 (avg A), bensolve 1.2. (avg B) and the Evans–Steuer algorithm (avg E) take to solve the problems. We also report the minimum (min A, min B, min E) and maximum (max A, max B, max E) running times for each algorithm among those 20 problems. The times are measured in seconds. The results are given in Table 3.

Table 3 Run time statistics for randomly generated degenerate problems where $q = 3$. For the first row $n = 5, m = 15$; for the second row $n = m = 10$; and for the last row $n = 15, m = 5$

min A	min B	min E	max A	max B	max E	avg A	avg B	avg E
0.02	0.01	0.03	0.14	0.09	140.69	0.07	0.04	8.85
0.03	0.02	0.08	1.33	0.14	4194.10	0.25	0.04	227.02
0.05	0.01	0.09	1.47	0.20	1893.00	0.05	0.25	190.25

Table 4 Computational results for single problems that require CPU times max A and max E among the ones that are generated for Table 3

$\|VS_A\|$	$\|VS_E\|$	$\|\bar{\mathcal{X}}_A\|$	$\|\bar{\mathcal{X}}_B\|$	$\|\bar{\mathcal{X}}_E\|$	$\|\bar{\mathcal{X}}^h_A\|$	$\|\bar{\mathcal{X}}^h_B\|$	$\|\bar{\mathcal{X}}^h_E\|$	time A	time B	time E
20	5617	1	1	1	0	0	0	0.13	0.03	140.69
30	361	3	3	3	0	0	0	0.14	0.06	6.61
324	22,871	1	1	4	0	0	1	1.33	0.03	4194.10
14	11,625	1	1	1	0	0	0	0.05	0.03	1893.00
452	11,550	1	1	1	39	2	4707	1.47	0.05	1598.10

For the first set of problems $n = 5, m = 15$; for the second set of problems $n = m = 10$ (max A and max E yielded the same problem here); and for the last set of problems $n = 15, m = 5$

Table 5 Run time statistics for randomly generated degenerate problems

q	n	m	min A	min B	max A	max B	avg A	avg B
4	10	30	0.05	0.03	177.47	4.07	8.49	0.21
4	20	20	0.21	0.02	973.53	199.77	19.89	12.05
4	30	10	0.11	0.03	2710.20	13.70	37.68	0.73

In order to give an idea how the solutions provided by the three algorithms differ for these degenerate problems, in Table 4 we provide detailed results for single problems. Among the 20 problems that are generated to obtain each row of Table 3, we select the two problems with the CPU times 'max A' and 'max E' and provide the following for them. $|\bar{\mathcal{X}}_{(\cdot)}|$ and $|\bar{\mathcal{X}}^h_{(\cdot)}|$ denote the number of elements of the set of point and direction maximizers that are found by each algorithm, respectively. $|VS_A|$ and $|VS_E|$ are the number of dictionaries that Algorithm 1 and the Evans–Steuer algorithm visit until termination. For each problem the time for Algorithm 1, bensolve 1.2, and the Evans Steuer algorithm to terminate are shown by 'time A', 'time B' and 'time E', respectively.

Finally, we compare Algorithm 1 and bensolve 1.2 to get statistical results regarding their efficiencies for degenerate problems. Note that this test was done on a different computer than the previous tests. Table 5 shows the numerical results for the randomly generated degenerate problems with $q = 4$ objectives, m constraints and n variables. We generate 100 problems for each size. We measure the average time that Algorithm 1 (avg A) and bensolve 1.2. (avg B) take to solve the problems. The minimum (min A, min B) and maximum (max A, max B) running times for each algorithm among those 100 problems are also provided.

Clearly, for degenerate problems bensolve 1.2 is more efficient than the simplex-type algorithms considered here, namely Algorithm 1 and the Evans–Steuer algorithm. However, the design of Algorithm 1 results in a significant decrease in CPU time compared to the Evans–Steuer algorithm in its improved form of [1,2].

Acknowledgements We would like to thank Andreas Löhne, Friedrich-Schiller-Universität Jena, for helpful remarks that greatly improved the manuscript, and Ralph E. Steuer, University of Georgia, for providing us the ADBASE implementation of the algorithm from [15]. Vanderbei's research was supported by the Office of Naval Research under Award Number N000141310093 and N000141612162.

References

1. Armand, P.: Finding all maximal efficient faces in multiobjective linear programming. Math. Program. **61**, 357–375 (1993)
2. Armand, P., Malivert, C.: Determination of the efficient set in multiobjective linear programming. J. Optim. Theory Appl. **70**, 467–489 (1991)
3. Barber, C.B., Dobkin, D.P., Huhdanpaa, H.T.: The Quickhull algorithm for convex hulls. ACM Trans. Math. Softw. **22**(4), 469–483 (1996)
4. Bencomo, M., Gutierrez, L., Ceberio, M.: Modified Fourier-Motzkin elimination algorithm for reducing systems of linear inequalities with unconstrained parameters. Departmental Technical Reports (CS) 593, University of Texas at El Paso (2011)
5. Benson, H.P.: 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**, 1–24 (1998)
6. Csirmaz, L.: Using multiobjective optimization to map the entropy region. Comput. Optim. Appl. **63**(1), 45–67 (2016)
7. Dauer, J.P.: Analysis of the objective space in multiple objective linear programming. J. Math. Anal. Appl. **126**(2), 579–593 (1987)
8. Dauer, J.P., Liu, Y.H.: Solving multiple objective linear programs in objective space. Eur. J. Oper. Res. **46**(3), 350–357 (1990)
9. Ecker, J.G., Hegner, N.S., Kouada, I.A.: Generating all maximal efficient faces for multiple objective linear programs. J. Optim. Theory Appl. **30**, 353–381 (1980)
10. Ecker, J.G., Kouada, I.A.: Finding all efficient extreme points for multiple objective linear programs. Math. Program. **14**(12), 249–261 (1978)
11. Ehrgott, M.: Multicriteria Optimization. Springer, Berlin (2005)
12. Ehrgott, M., Löhne, A., Shao, L.: A dual variant of Benson's outer approximation algorithm. J. Global Optim. **52**(4), 757–778 (2012)
13. Ehrgott, M., Puerto, J., Rodriguez-Chía, A.M.: Primal-dual simplex method for multiobjective linear programming. J. Optim. Theory Appl. **134**, 483–497 (2007)
14. Ehrgott, M., Shao, L., Schöbel, A.: An approximation algorithm for convex multi-objective programming problems. J. Global Optim. **50**(3), 397–416 (2011)
15. Evans, J.P., Steuer, R.E.: A revised simplex method for multiple objective programs. Math. Program. **5**(1), 54–72 (1973)
16. Hamel, A.H., Löhne, A., Rudloff, B.: Benson type algorithms for linear vector optimization and applications. J. Global Optim. **59**(4), 811–836 (2014)
17. Isermann, H.: The enumeration of the set of all efficient solutions for a linear multiple objective program. Oper. Res. Q. **28**(3), 711–725 (1977)
18. Kalyanasundaram, B., Pruhs, K.R.: Constructing competetive tours from local information. Theor. Comput. Sci. **130**, 125–138 (1994)
19. Löhne, A.: Vector Optimization with Infimum and Supremum. Springer, Berlin (2011)
20. Löhne, A.: BENSOLVE: A free VLP solver, version 1.2. (2012). http://ito.mathematik.uni-halle.de/~loehne

21. Löhne, A., Rudloff, B., Ulus, F.: Primal and dual approximation algorithms for convex vector optimization problems. J. Global Optim. **60**(4), 713–736 (2014)
22. Löhne, A., Weißing, B.: BENSOLVE: A free VLP solver, version 2.0.1 (2015). http://bensolve.org/
23. Löhne, A., Weißing, B.: The vector linear program solver bensolve: notes on theoretical background. Eur. J. Oper. Res. (2016). doi:10.1016/j.ejor.2016.02.039
24. Luc, D.: Theory of Vector Optimization, Lecture Notes in Economics and Mathematical Systems, vol. 319. Springer, Berlin (1989)
25. Makhorin, A.: GLPK (GNU linear programming kit) (2012). https://www.gnu.org/software/glpk/
26. Przybylski, A., Gandibleux, X., Ehrgott, M.: A recursive algorithm for finding all nondominated extreme points in the outcome set of a multiobjective integer programme. INFORMS J. Comput. **22**, 371–386 (2010)
27. Ruszczyński, A., Vanderbei, R.J.: Frontiers of stochastically nondominated portfolios. Econometrica **71**(4), 1287–1297 (2003)
28. Schechter, M., Steuer, R.E.: A correction to the connectedness of the Evans–Steuer algorithm of multiple objective linear programming. Found. Comput. Decis. Sci. **30**(4), 351–359 (2005)
29. Shao, L., Ehrgott, M.: Approximately solving multiobjective linear programmes in objective space and an application in radiotherapy treatment planning. Math. Methods Oper. Res. **68**(2), 257–276 (2008)
30. Shao, L., Ehrgott, M.: Approximating the nondominated set of an MOLP by approximately solving its dual problem. Math. Methods Oper. Res. **68**(3), 469–492 (2008)
31. Steuer, R.E.: A Multiple Objective Linear Programming Solver for All Efficient Extreme Points and All Unbounded Efficient Edges. Terry College of Business, University of Georgia, Athens (2004)
32. Vanderbei, R.J.: Linear Programming: Foundations and Extensions. Kluwer Academic Publishers, Dordrecht (2013)
33. Zionts, S., Wallenius, J.: Identifying efficient vectors: some theory and computational results. Oper. Res. **28**(3), 786–793 (1980)