ALGORITHMS FOR ON-LINE VERTEX ENUMERATION PROBLEM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

By
İrfan Caner KAYA
September 2017

ALGORITHMS FOR ON-LINE VERTEX ENUMERATION PROBLEM

By İrfan Caner KAYA

September 2017

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Firdevs ULUS(Advisor)

Özlem Karsu

Banu Lokman

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

ALGORITHMS FOR ON-LINE VERTEX ENUMERATION PROBLEM

İrfan Caner KAYA
M.S. in Industrial Engineering
Advisor: Firdevs ULUS
September 2017

Vertex enumeration problem is to enumerate all vertices of a polyhedron P which is given by intersection of finitely many halfspaces. It is a basis for many algorithms designed to solve problems from various application areas and there are many algorithms to solve these problems in the literature. On the one hand, there are iterative algorithms which solve the so called 'on-line' vertex enumeration problem in each iteration. In other words, in each iteration of these algorithms, the current polyhedron is intersected with an additional halfspace that defines P. On the other hand, there are simplex-type algorithms which takes the set off all halfspaces as its input from the beginning.

One of the usages of the vertex enumeration problem is the 'Benson-type' multiobjective optimization algorithms. The aim of these algorithms is to generate or approximate the Pareto frontier (the set of nondominated points in the objective space). In each iteration of the Benson's algorithm, a polyhedron which contains the Pareto frontier is intersected with an additional halfspace in order to find a finer outer approximation. The vertex enumeration problem to be used within this algorithm has a special structure. Namely, the polyhedron to be generated is known to be unbounded with a recession cone which is equal to the positive orthant.

In this thesis, we consider the 'double description method' which is a method to solve an on-line vertex enumeration problem where the starting polyhedron is bounded. (1) We generate an iterative algorithm to solve the vertex enumeration problem from the scratch where polyhedron P is allowed to be bounded or unbounded. (2) Then, we slightly modify this algorithm to be more efficient while it only works for problems where the recession cone of P is known to be

the positive orthant. (3) Finally, we generate an additional algorithm for these problems. For this one, we modify the double description method such that it uses the extreme directions of the recession cone more effectively. We provide an illustrative example to explain the algorithms in detail.

We implement the algorithms using MATLAB; employ each of them as a function of a 'Benson-type' multiobjective optimization algorithm; and test the performances of the algorithms for randomly generated linear multiobjective optimization problems. Accordingly, for two dimensional problems, there is no clear distinction between the run-time performances of these algorithms. However, as the dimension of the vertex enumeration problem increases, the last algorithm (Algorithm 3) gets more efficient than the others.

Keywords: vertex enumeration, on-line vertex enumeration, algorithms, multiobjective optimization.

ÖZET

ÇEVRİMİÇİ KÖŞE NOKTASI PROBLEMİ İÇİN ALGORİTMALAR

İrfan Caner KAYA
Endüstri Mühendisliği, Yüksek Lisans
Tez Danışmanı: Firdevs ULUS
Eylül 2017

Köşe noktası sayma problemi, sonlu sayıda yarı uzayın kesişimi olarak verilmiş bir çokyüzlü P'nin bütün köşe noktalarını bulmaktır. Bu problem, çeşitli uygulama alanlarında karşılaşılan problemleri çözmek için tasarlanmış birçok algoritmanın temelini oluşturmaktadır ve bu problemleri çözmek için literatürde çok sayıda algoritma mevcuttur. Bir yanda, her yinelemede, çevrimiçi köşe noktası sayma problemi adı verilen problemleri çözen yinelemeli algoritmalar vardır. Başka bir deyişle, bu algoritmaların her yinelemesinde, şu anki çokyüzlü, P'yi tanımlayan başka bir yarı uzayla kesiştirilir. Diğer yanda ise, bütün yarı uzayları başlangıçtan itibaren girdi olarak alan simpleks türü algoritmalar vardır.

Köşe noktası sayma probleminin kullanımlarından biri 'Benson'-türü çok amaçlı optimizasyon algoritmalarıdır. Bu algoritmaların amacı Pareto sınırına (amaç uzayında baskın olmayan noktalar kümesine) ulaşmak veya yaklaşmaktır. Benson Algoritması'nın her yinelemesinde, daha iyi bir dış yaklaşım bulmak için, Pareto sınırını içeren bir çokyüzlü ek bir yarı uzay ile kesiştirilir. Bu algoritma içinde kullanılan köşe noktası sayma probleminin özel bir yapısı vardır. Şöyle ki, bulunmak istenen çokyüzlünün, resesyon konisi pozitif ortant (orthant) olan sınırsız bir çokyüzlü olduğu bilinmektedir.

Bu tezde, başlangıç çokyüzlüsü sınırlı olan bir çevrimiçi köşe noktası sayma problemini çözmek için kullanılan 'çift betimleme (double description)' metodunu göz önünde bulundurduk. (1) Sınırlı ya da sınırsız olması mümkün çokyüzlü P için köşe noktası sayma problemini en baştan başlayarak çözen yinelemeli bir algoritma oluşturduk. (2) Daha sonra, bu algoritmayı, sadece resesyon konisi pozitif ortant olan P için ve daha etkin çalışacak bir şekilde modifiye ettik. (3) Son olarak, bu problemlere yönelik ek bir algoritma oluşturduk. Bu algoritma için,

çift betimleme yöntemini resesyon konisinin aşırı yönlerini daha etkin olarak kullanacak şekilde modifiye ettik. Bu algoritmaları detaylı bir şekilde anlatabilmek için açıklayıcı bir örnek sunduk.

Bu algoritmaları MATLAB kullanarak uygulamaya geçirdik; her bir algoritmayı 'Benson'-türü çok amaçlı bir optimizasyon algoritmasının içinde fonksiyon olarak kullandık ve rasgele oluşturulan doğrusal çok amaçlı optimizasyon problemleri için algoritmaların performanslarını test ettik. Bu doğrultuda, iki boyutlu problemler için algoritmaların çalışma süresi performanslarında bir fark yoktur. Ancak, problemlerin boyutu büyüdükçe son algoritma (Algoritma 3) diğerlerine göre daha verimli hale gelir.

Anahtar sözcükler: köşe noktası sayma, çevrimiçi köşe noktası sayma, algoritmalar, çok amaçlı optimizasyon.

Acknowledgement

I would like to express my sincere gratitude to my advisor Asst. Prof. Firdevs Ulus to support my study. She expanded my perspective to solve a problem in a different way. Her knowledge and insights contributed me to algorithm development and improve its performance. Moreover, her comments are always driving force to seek the solutions of the problem and she is a good mentor.

Also, I would like to thank my family for supporting and understanding me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A polyhedron P can be represented as intersection of finitely many halfspaces or as convex hull of its vertices added to the conic hull of its extreme directions. The problem of computing the vertex representation of P from its halfspace representation is called the *vertex enumeration problem*. On the other hand, computing the halfspace representation of P from its vertex representation is called the *face enumeration problem*. The vertex enumeration problem has been studied for many years, starting from the 1930s. The main focus of many early algorithms was to find the respective representations correctly and accurately, see for instance [1, 2, 3, 4]. However, enumerating vertices/faces of a polyhedron in general is a computationally difficult task ([5, 6]) and since 1980s the main focus has been to improve the computational efficiency of the existing algorithms or to propose new competitive ones, see [7, 8, 9, 10].

The problem of computing the vertices of a polyhedron that is given as the intersection of finitely many halfspaces is sometimes called the *off-line* vertex enumeration problem. On the other hand, finding the vertices of a polyhedron P'' given as intersection of a single halfspace H and another polyhedron P' whose vertex representation is known is called the *on-line* vertex enumeration problem.

Note that an on-line vertex enumeration algorithm can be called repetitively in order to solve an off-line vertex enumeration problem. However, there are also algorithms that solve the off-line vertex enumeration problems directly. These are mainly (simplex-type) pivoting algorithms, see for instance [11, 12].

The problem of finding vertices of a polyhedron is fundamental and it is the base of many other algorithms including the simplex algorithm to solve linear programs. Recently, it has also been an important part of many algorithms designed to solve multiobjective optimization problems, [13, 14, 15, 16]. In general, these 'Benson-type' multiobjective optimization algorithms aim to find a polyhedral approximation to the Pareto frontier in the objective space. In order to obtain this, an on-line vertex enumeration problem has to be solved in each iteration of these algorithms. Indeed, the most time consuming subroutine of these algorithms is the vertex enumeration part [15].

Many non-pivoting on-line vertex enumeration algorithms in the literature are designed to find the vertices of 'bounded' polyhedrons, that is, polyhedrons with no recession directions, see for instance [17]. However, for Benson-type multiobjective optimization algorithms, one needs to find the vertices of 'unbounded' polyhedrons since (the approximation of) the Pareto frontier is the boundary of an unbounded polyhedron whose recession cone is the positive orthant.

In this thesis, the aim is to come up with efficient vertex enumeration algorithms in order to employ it as a subroutine in Benson-type multiobjective optimization algorithms.

We first consider the 'double description' method, which is designed to solve the on-line vertex enumeration problem, as it is described in [16]. This method takes a polyhedron $P^{'}$ and a halfspace H as its inputs; assumes that the vertices of $P^{'}$ is given; and returns the vertices of the polyhedron $P^{''} = P^{'} \cap H$.

Then, we propose Algorithm 1, which solves the 'off-line' vertex enumeration problem by using the double description method in each iteration. This algorithm starts with the halfspace representation of a polyhedron and returns the set of

all vertices and, if the polyhedron is unbounded, the set of all extreme directions of it. The main idea is to start with a sufficiently large bounded polyhedron which contains the set of all vertices and to iterate until all the halfspaces are intersected. Algorithm 1 works for general problems and it does not utilize the recession cone information of the unbounded polyhedron that is given in Benson-type multiobjective optimization algorithms. Hence, we slightly modify this and propose Algorithm 2.

Algorithm 2 assumes that the recession cone of polyhedron P is the positive orthant and it also assumes that an initial vertex v^0 such that $v^0 + \mathbb{R}^n_+ \supseteq P$ is known. The idea of it is very similar to Algorithm 1 and the main difference is the initialization part.

Next, in order to come up with an algorithm which uses the extreme direction information even more directly and efficiently, we modify the 'double description' method such that it works with unbounded polyhedrons whose recession cone is the positive orthant. Then, we propose Algorithm 3 which starts with an initial vertex v^0 as in Algorithm 2; and iterates using this modified double description method.

We use MATLAB software to implement the algorithms. In order to compare the efficiencies, we call them within a MATLAB implementation of a Benson-type convex multiobjective optimization solver, called c-bensolve. Note that bensolve is a linear multi-objective optimization solver, see [18] and c-bensolve is a MATLAB implementation of an extension of it to the convex programs [19].

Chapter 2 provides a literature review of vertex enumeration problems. The literature on pivoting and non-pivoting (or iterative) algorithms are discussed separately. In Chapter 3, we provide preliminaries on basic convex analysis and on multiobjective optimization problems. The algorithms are described in Chapter 4. We provide the pseudo-codes of all the algorithms and necessary subroutines. Moreover, an illustrative example to explain the algorithms in detail is provided. Chapter 5 presents the computational results of the algorithms. We conclude and discuss the future work of our study in Chapter 6.

Chapter 2

Literature Review

Over the years, there have been several methods proposed to solve the vertex enumeration problem. In 1980, Mattheiss and Rubin [20] and in 1983, Dyer [21] conduct surveys regarding these algorithms. As both of these surveys assert, there are two classes of vertex enumeration algorithms. On the one hand, some of the algorithms are based on pivoting methods and they use the Dantzig's simplex tableau [22]. On the other hand, there are non-pivoting algorithms which are based on the double description method developed by Motzkin et al [23]. These methods, in general, are based on the geometrical features of the problem. These are also known as 'constructive-inductive' or 'iterative' methods.

2.1 Pivoting Algorithms

There are many pivoting vertex enumeration algorithms which are based on the Dantzig's simplex tableau.

In 1961, Balinski [11] proposes a variant of simplex method that finds all vertices of a given convex polyhedral set. Indeed, it finds all basic solutions of a

linear program rather than finding a single optimal solution as Dantzig's simplex method does. Another simplex-type algorithm is proposed by Maňas and Nedoma in 1968 [24]. It finds the set of all vertices of a polyhedron and it is based on the theory of graphs. In 1975, Altherr [12] develops an algorithm that is based on [24].

Another pivoting vertex enumeration algorithm is proposed by Mattheiss [25] in 1973. Different from the previous methods, this algorithm detects the redundant constraints in an efficient way. It embeds the given bounded convex polyhedron to one-higher dimensional Euclidean space and constructs a connected graph which is used to eliminate the redundant constraints. Then, the simplex tableau is used to generate the set of vertices. In 1980, Mattheiss and Schmidt [8] analyse the computational efficiency of Mattheiss's Algorithm [25] and compare it to the algorithm proposed by Klee in 1974 [26]. Accordingly, Mattheiss's Algorithm outperforms the algorithm by Klee.

In 1977, Dyer and Proll [27] develop an algorithm based on the simplex tableau which works for bounded convex polyhedral sets. Rather than employing the simplex tableau and pivoting 'unnecessary' elements, this algorithm creates a spanning tree in the graph induced by the given polyhedron and make some comparisons in order to check adjacency. In 1982, Dyer and Proll [7] improve this algorithm to make it computationally more efficient.

Avis and Fukuda [28] propose a new pivot-based algorithm in 1992. This algorithm does not need to use data structures for its implementation and it also works better if there is degeneracy. This algorithm is utilized to solve three different problems related to computational geometry: (1) enumeration of facets of a polytope given as the convex hull of a finite set of points, (2) enumeration of vertices of a convex polyhedron given as intersection of finitely many halfspaces and (3) enumeration of intersection points of a finite set of hyperplanes. Avis [9] improves the computational efficiency of this algorithm by employing the reverse search technique.

In 1994, Provan [29] develops an algorithm to enumerate all vertices of a convex polyhedron which does not have to be bounded. It has been shown that the running time of the algorithm is polynomial with respect to the number of extreme points. In 1995, Ottmann, Schuierer and Soundaralakshmi [30] propose an algorithm that enumerates the vertices of polyhedrons that are given in higher dimensions, $d \geq 4$.

Fukuda, Liebling and Margot [31] propose a backtracking algorithm for solving vertex and face enumeration problems in 1997. In this paper, it is underlined that the backtracking algorithm is efficient in the sense of time complexity and requiring less data storage. In 1998, Bremner, Fukuda and Marzetta [32] develop primal and dual algorithms that solve the vertex enumeration and the face enumeration problems using the duality relation between the two.

2.2 Non-Pivoting Algorithms

Different from pivoting algorithms, there are methods which utilize the geometrical features of the vertex enumeration problem. In general, these algorithms use the double description method developed by Motzkin et al. [23] and solve the problem iteratively.

In 1960's Chernikova [3, 4, 33] proposes algorithms for calculating the non-negative solutions of a system of linear equations and inequalities. These algorithms firstly find the extreme directions of the solution set and then calculate the extreme points. Much later, in 1994, H. Le Verge [34] extends Chernikova's algorithms and discusses the issues regarding the implementation of these algorithms. It also points out that Chernikova's Algorithm can be applied to the dual problem which is finding non-redundant set of facet supporting hyperplanes.

In 1975, Greenberg [1] proposes an algorithm to find the set of all vertices of a given convex polyhedral set. The algorithm is based on the theoretical results provided by Uzawa [35] and Stoke [36]. In 1976, Sherman [37] demonstrates that

Greenberg's algorithm can find extraneous solutions in case of degeneracy. Later, Dyer and Proll [38] develop a different variant of Greenberg's Algorithm which works better for degenerate problems.

In 1991, Chen, Hansen and Jaumard [17] propose on-line and off-line vertex enumeration algorithms by using adjacency lists of the vertices. The algorithms work for bounded polyhedrons but the extensions to the unbounded cases are also explained.

Note that the face enumeration problem is equivalent to the vertex enumeration problem as they are dual to each other. There are also methods that are designed to solve the face enumeration problem directly, see for instance [39, 40, 41].

We exploit the double description method [10] that is used by Czirmaz[16] to solve vertex enumeration problem within Benson's algorithm for developing Algorithm 1, Algorithm 2 and Algorithm 3. In addition, we directly use extreme directions information to solve vertex enumeration problem for Algorithm 3.

Chapter 3

Preliminaries

A hyperplane $H \subseteq \mathbb{R}^n$ is given by

$$H = \{x \in \mathbb{R}^n \mid a^T x = b\}$$

for some $a \in \mathbb{R}^n \setminus \{0\}$ and $b \in \mathbb{R}$. A halfspace is one of the two regions into which a hyperplane divides $\mathbb{R}^n$. In particular we denote the two halfspaces given by H as follows:

$$H^- = \{x \in \mathbb{R}^n \mid a^T x \leq b\} \text{ and } H^+ = \{x \in \mathbb{R}^n \mid a^T x \geq b\}.$$

A subset $S \subseteq \mathbb{R}^n$ is called *convex* if $\lambda x + (1-\lambda)y \in S$, for all $x, y \in S$ and $0 \leq \lambda \leq 1$. Let $x, y \in \mathbb{R}^n$. If $z \in \mathbb{R}^n$ can be written as $z = \lambda x + (1-\lambda)y$ for some $\lambda \in [0,1]$, then it is called a *convex combination* of x and y. Moreover, if λ is not equal to 0 or 1, then it is called a *strict convex combination* of x and y.

The smallest convex set that contains S is called the *convex hull* of S and it is denoted by $\operatorname{conv} S$. For a finite set F, namely, $F := \{x_1, x_2, \ldots, x_m\}$ where $m \in \mathbb{N}$, we have

$$\operatorname{conv} F = \{\sum_{i=1}^{m} \lambda_i x_i \mid \sum_{i=1}^{m} \lambda_i = 1, \lambda_i \geq 0, i = 1, \ldots, m\}.$$

For a convex set $S \subseteq \mathbb{R}^n$, $x \in S$ is said to be an *extreme point* or a *vertex* of S if $x = \lambda y + (1-\lambda)z$ for $y, z \in S$ and $\lambda \in (0,1)$ implies that $x = y = z$. Let $F \subseteq S$ be a convex subset of S and $x, y \in S$ be arbitrary. If $\lambda x + (1-\lambda)y \in F$ for some $0 < \lambda < 1$ holds only if both x and y are elements of F, then F is a *face* of S. A line segment between two extreme points is said to be an *edge* of S if it in on a face of S.

A subset $C \subseteq \mathbb{R}^n$ is called *cone*, if $\gamma x \in C$, $\forall x \in C$ and $\gamma \geq 0$. If C is also a convex set, then it is called a *convex cone*. A convex cone C is called *pointed* if it does not possess any line, or equivalently, if the origin is the only extreme point of C. If C can be represented as:

$$C = \{x \in \mathbb{R}^n \mid Ax \geq 0\}$$

for some matrix $A \in \mathbb{R}^{n \times m}$, then it is called *polyhedral cone*. Note that C can also be defined as the intersection of finitely many halfspaces that support the origin in $\mathbb{R}^n$.

The *conic hull* of set $S \subseteq \mathbb{R}^n$ is the smallest cone that contains S, and it is denoted by $\operatorname{cone} S$. For a finite set F given by $F := \{x_1, x_2, \ldots, x_m\}$ where $m \in \mathbb{N}$, we have

$$\operatorname{cone} F = \{\sum_{i=1}^{m} \lambda_i x_i \mid \lambda_i \geq 0, i = 1, \ldots, m\}.$$

For a subset S of $\mathbb{R}^n$, $d \in \mathbb{R}^n \setminus \{0\}$ is a *recession direction* (or simply *direction*) of S, if $x + \gamma d \in S$ for all $\gamma \geq 0$ and $x \in S$. The set of all recession directions constitute the *recession cone* of S which is denoted by $\operatorname{recc} S$. Note that we have

$$\operatorname{recc} S = \{d \in \mathbb{R}^n \mid x + \gamma d \in S \ \text{ for all } \ \gamma \geq 0,\ x \in S\}.$$

A recession direction $d \in \operatorname{recc} S \setminus \{0\}$ of convex set S is said to be an *extreme direction* of S if $d = \lambda_1 d_1 + \lambda_2 d_2$ for some $d_1, d_2 \in \operatorname{recc} S \setminus \{0\}$ and $\lambda_1, \lambda_2 > 0$ implies that $\frac{d}{\|d\|} = \frac{d_1}{\|d_1\|} = \frac{d_2}{\|d_2\|}$. $S \subseteq \mathbb{R}^n$ is called *bounded* if $\operatorname{recc} S = \{0\}$.

If a convex set P can be written as

$$P = \{x \in \mathbb{R}^n \mid A^T x \geq b\}, \tag{3.1}$$

where $A \in \mathbb{R}^{n\times m}$ and $b \in \mathbb{R}^m$, then it is called a *polyhedral convex set* or a *convex polyhedron*. A bounded polyhedron is sometimes called a *polytope*.

Note that P given by (3.1) is intersection of finitely many half-spaces, namely,

$$P = \bigcap_{i=1}^{m} \{x \in \mathbb{R}^n \mid a_i^T x \geq b_i\}, \tag{3.2}$$

where $a_i \in \mathbb{R}^n$ is the i^{th} column of matrix A and b_i is the i^{th} component of b. The representation given in (3.2) is called *H-representation* or *halfspace representation* of P. On the other hand, if P given by (3.1) is non-empty, it can also be represented as the convex hull of all its extreme points added to the conic hull of all its extreme directions. To be more precise, let $V = \{v_1, v_2, ..., v_k\}$ be the set of all extreme points of P and $D = \{d_1, d_2, ..., d_l\}$ be the set of all extreme directions of P, where $k, l \in \mathbb{N}$. Then, P can be written as

$$P = \{x \in \mathbb{R}^n \mid x = \sum_{i=1}^{k} \lambda_i v_i \ + \ \sum_{j=1}^{l} \gamma_j d_j, \lambda_i \geq 0, \gamma_j \geq 0, \sum_{i=1}^{k} \lambda_i = 1\}. \tag{3.3}$$

Clearly, $k = 0$ means that P has no vertex, for instance it can be a halfspace; and $l = 0$ means that P is bounded. The representation given by (3.3) of P is called the *V-representation* or the *vertex representation* of the polyhedral convex set P. The problem of finding the *V-representation* of a set given its *H-representation* is called the *vertex enumeration problem*. On the other hand, the problem of finding the *V-representation* of a set given its *H-representation* is called the *face enumeration problem*.

3.1 Convex Multiobjective Optimization and Benson-type Algorithms

A convex multiobjective optimization problem (MOP) is given by

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

where $F(x) = [f_1(x), \ldots, f_n(x)]^T$ with $f_i : \mathbb{R}^N \to \mathbb{R}$ for all $i = 1, \ldots, n$ are convex functions and the feasible set $X \subseteq \mathbb{R}^N$ is a convex set. The image of the feasible set is given by $F(X) = \{F(x) \in \mathbb{R}^n \mid x \in X\}$ and the set

$$\mathcal{P} := \mathrm{cl}\,(F(X) + \mathbb{R}^n_+) \tag{3.4}$$

is called the *upper image* of (P). It is known that $\mathcal{P}$ is convex and closed.

The *ideal point* of problem P can be found by minimizing f_i, for $i = 1, \ldots, n$ over feasible set X. More specifically, let $z_i := \min\{f_i(x) \mid x \in X\}$. Then, $z^I := [z_1, \ldots, z_n]^T$ is the ideal point of (P).

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

$$(\{y\} - \mathbb{R}^n_+) \cap F(X) = \{y\},$$

where the minus sign stands for the Minkowski (point-wise) subtraction. Similarly $y \in F(X)$ is said to be a *weakly minimal* or a *weakly non-dominated point* if

$$(\{y\} - \mathrm{int}\,\mathbb{R}^n_+) \cap F(X) = \emptyset,$$

where $\mathrm{int}\,\mathbb{R}^n_+$ is the interior of the positive orthant in $\mathbb{R}^n$. A feasible point $x \in X$ is said to be a *(weak) minimizer* or a *(weakly) efficient solution* if $F(x)$ is a (weakly) non-dominated point in $F(X)$.

In some applications of multiobjective optimization it may be enough to generate some non-dominated points of $F(X)$, respectively some efficient solutions. In order to find a single (weakly) efficient solution, one generally solves a single objective optimization problem, which is called a *scalarization*. There are many different scalarization techniques in the literature. In some other applications of MOP, it is important to generate the set of all (weakly) non-dominated points of $F(X)$, respectively the set of all efficient solutions.

If the problem is linear, $\mathcal{P}$ is a polyhedral set and it is possible to 'solve' the problem in the sense that it is possible to generate the set of all efficient solutions.

There are different approaches to 'solve' linear MOPs in the literature. On the one hand, there are simplex-type algorithms which work in the variable space, see for instance [42, 43, 44]. On the other hand, there are Benson-type algorithms which work in the objective space, [13, 14, 16]. If the problem is nonlinear convex, it is not possible to generate the set of all (weakly) non-dominated points in general. Instead, 'Benson-type' objective space algorithms can generate approximations to the set of all weakly non-dominated points [15, 45].

As mentioned above, Benson-type MOP algorithms work in the objective space and aim to find or approximate the upper image $\mathcal{P}$ since the boundary of the upper image, $\operatorname{bd}\mathcal{P}$, contains the set of all weakly non-dominated points in the image of the feasible set.

The general idea of the algorithm can be described as follows. It starts with a point v^0 in the objective space which satisfies $v^0 + \mathbb{R}^n_+ \supseteq \mathcal{P}$. Clearly, the ideal point z^I of problem (P) can be taken as v^0. Then, the initial outer approximation of $\mathcal{P}$ is $P^0 := v^0 + \mathbb{R}^n_+$. For the first iteration, a Benson-type scalarization is solved depending on v^0. The idea is to find point y^0 on $\operatorname{bd}\mathcal{P}$ that is closest (in some sense) to v^0. The dual solution of the scalarization also yields the supporting hyperplane of $\mathcal{P}$ at y^0. Then, P^1 is the intersection of P^0 and the halfspace that supports P^0 at y^0.

At k^{th} iteration of the algorithm, the first step is to find the vertices of the current outer approximation P^k. Then, for a vertex v_k of P^k, a Benson-type scalarization is solved in order to find a (weakly) efficient solution x_k, the corresponding (weakly) non-dominated point $y_k \in \operatorname{bd}\mathcal{P}$ and a supporting hyperplane H_k of $\mathcal{P}$ at y_k. Then the current outer approximation is updated as $P^{k+1} = P^k \cap H_k$.

The algorithm stops when all the vertices of the current outer approximation are in ϵ-distance to the upper image, where ϵ is the approximation error given by the user.

Remark 3.1.1. *Note that in each iteration of these algorithms a vertex enumeration problem is solved. Indeed, these vertex enumeration problems are in a special form, namely, the polyhedron to be found is known to be unbounded since $\mathcal{P}$ is an*

unbounded set. Moreover, the recession cone of the polyhedron is known and it is $\mathbb{R}^n_+$*. Hence the extreme directions of the recession cone is nothing but the unit directions,* $e_1, \ldots, e_n$*.*

Chapter 4

Algorithms

In this chapter, we describe some vertex enumeration algorithms in the literature and the modified algorithms that we propose in order to use in Benson-type MOP algorithms. We assume that an H-representation of polyhedron P is known and the aim is to find a V-representation of it.

In order to explain the algorithms, first the double description method as given in [16] is described. This method finds the vertices of polyhedron $P'' := P' \cap H$, where H is an halfspace and P' is a bounded polyhedron whose V-representation is given. This is also known as an on-line vertex enumeration procedure and it can be used iteratively in order to solve a vertex enumeration problem from scratch.

Next, we describe different algorithms that use the double description method in order to solve the vertex enumeration problem. The first algorithm (Algorithm 1) works for the problems where polyhedron P is allowed to be bounded or unbounded. The algorithm starts with a 'sufficiently large' artificial initial polyhedron and iterates by intersecting the current polyhedron with an additional halfspace until all the halfspaces that define polyhedron P are intersected. Note that if P is unbounded and the extreme directions of the recession cone of P are not known, they can also be computed by the algorithm. However, since for many applications, the only concern is to find the vertices and since finding the

extreme directions would require some additional computational time, we keep this as an optional step.

The second and the third algorithms are modifications of the first one where we assume that P is unbounded and the extreme directions of its recession cone are known from the beginning. The motivation for this assumption is that we aim to use these algorithms as a subroutine in Benson-type multiobjective optimization algorithms, see Remark 3.1.1.

The only difference between the second algorithm (Algortihm 2) and the first one is in their initialization steps. The third algorithm (Algorithm 3) is also similar to the first two but instead of starting with an artificial bounded polyhedron we initialize it by an unbounded one and directly use extreme directions in the computations of the updated polyhedrons.

In the last section of this Chapter, we provide an illustrative example to explain the algorithms in detail.

4.1 The Double Description Method

Let $P^{'}$ be a bounded convex polyhedron, that is $\operatorname{recc} P^{'} = \{0\}$, and let H be a halfspace given by $H := \{x \in \mathbb{R}^n \mid a^T x \geq b\}$ for some $a \in \mathbb{R}^n \setminus \{0\}$ and $b \in \mathbb{R}$. For simplicity, the hyperplane given by the boundary of H is denoted by h.

In order to compute a V- *representation* of the polyhedron $P^{''} = P^{'} \cap H$, we use the double description method from [16].

Let $V^{'}$ be the set of vertices and $F^{'}$ be the set of faces of $P^{'}$. The following is a useful definition in order to describe the method.

Definition 4.1.1. *If vertex $v \in V^{'}$ is on face $f \in F^{'}$; then v is said to be an* adjacent vertex *of face f and f is said to be an* adjacent face *of vertex v.*

For a vertex v, let F_v denote the set of all adjacent faces of v, and for a face

f, let V_f denote the set of all adjacent vertices of f. We call these the adjacency lists. It is assumed that the sets $V^{'}, F^{'}$ as well as V_f, F_v for all $v \in V^{'}$ and $f \in F^{'}$ are known.

As it is an important subroutine in the double description method, we first describe a procedure to check if there is an edge between given two vertices of polyhedron $P^{'}$. Let $v^+, v^- \in V^{'}$ be two vertices. In order to check if there is an edge between them, we consider the set of faces which are both adjacent to v^+ and v^-. Then, for each face in this set, we consider the adjacent vertices of it. If the intersection of the set of vertices over all these faces consists of only v^+ and v^- then, there is an edge between the two; otherwise, there is no edge between them. One can check [16] for the details of the method. Procedure 1 is the pseudo-code for the *isedge* function, which takes polyhedron $P^{'}$ and vertices v^+, v^- as its input; and returns the set of faces that contains the edge between them if there is any or returns empty set otherwise. Note that with $P^{'}$ as an input we mean that *isedge* has the access to $V^{'}, F^{'}$ as well as V_f and F_v for all $f \in F^{'}$ and $v \in V^{'}$.

Procedure 1 $isedge(P^{'}, v^+, v^-)$

Let $F^{\pm} := F_{v^+} \cap F_{v^-}$;
Let $V^{\pm} := V^{'}$;
for $i = 1 : |F^{\pm}|$ **do**
    Let f^i be the i^{th} face in $F^{\pm}$;
    $V^{\pm} \leftarrow V^{\pm} \cap V_{f^i}$;
end for
if $V^{\pm} = \{v^+, v^-\}$ **then**
    return $F^{\pm}$
else
    return $\emptyset$
end if

The idea of the double description method is as follows: First, it checks if each vertex v in $V^{'}$ is in the interior of H, on the boundary h of H, or not included in H. Clearly, the ones that are not in H will not be a vertex of the updated polyhedron anymore. As long as there exist at least one vertex that is included in H and there exists at least one vertex that is not included in it then, the algorithm considers each couple of vertices v^+ and v^- in $V^{'}$, where $v^+ \in \operatorname{int} H$ and $v^- \notin H$,

and checks if there is an edge between v^+ and v^-, using *isedge* subroutine. For the couples that form an edge, a new vertex is found by intersecting the edge with hyperplane h. This new vertex is a vertex of the updated polyhedron $P^{''}$ and h is a face of $P^{''}$. Each time a new vertex is found, the adjacency lists are updated accordingly.

The pseudo-code for the double description method is given by Procedure 2. The function *onlinevert* takes polyhedron $P^{'}$ ($V^{'}, F^{'}, V_f, F_v$ for all f, v) and halfspace H as its input. The first step is to determine if each vertex of the given polyhedron is in the interior of H; on the boundary of H or not included in H. Accordingly, three respective subsets V^+, V^0, and V^- are formed, see Procedure 2 (Part 1). On the one extreme, if none of the vertices are included in H, then $P^{''}$ is empty set; and on the other extreme if all the vertices are in H, then H is a redundant halfspace in defining $P^{''}$, hence, $P^{''} = P^{'}$, see Procedure 2 (Part 2) lines 1-4.

Procedure 2 (Part 1) *onlinevert*$(P^{'}, H)$

```
1: Initialize V^0, V^+, V^- := ∅;
2: for all v ∈ V' do
3:    if a^T v > b (i.e. v ∈ int H) then
4:       V^+ ← V^+ ∪ {v};
5:    else if a^T v = b (i.e. v ∈ h) then
6:       V^0 ← V^0 ∪ {v};
7:    else if a^T v < b (i.e. v ∉ H) then
8:       V^- ← V^- ∪ {v};
9:    end if
10: end for
```

In case there exists at least one vertex within H and one vertex out of it; then hyperplane h would be a face of the updated polyhedron, so h is added to $F^{'}$, see Procedure 2 (Part 2) line 6. The algorithm continues by checking the vertices that are on hyperplane h, that is, the vertices in V^0. Note that $v \in V^0$ is still a vertex of the updated polyhedron. Moreover, by definition, each $v \in V^0$ is an adjacent vertex of this new face h and similarly, h is an adjacent face of each of these vertices (line 8).

The next step is to find the new vertices that are formed by the intersection of

Procedure 2 (Part 2) *onlinevert*$(P^{'}, H)$

1: **if** $V^{+} \cup V^{0} = V^{'}$ **then**
2: **return** $P^{''} = P^{'}$;
3: **else if** $V^{-} = V^{'}$ **then**
4: **return** $P^{''} = \emptyset$;
5: **else**
6: $F^{'} \leftarrow F^{'} \cup \{h\}$, $V_h = \emptyset$;
7: **for all** $v \in V^{0}$ **do**
8: $V_h \leftarrow V_h \cup \{v\}$ and $F_v \leftarrow F_v \cup \{h\}$;
9: **end for**
10: **for all** $v^{+} \in V^{+}$ **do**
11: **for all** $v^{-} \in V^{-}$ **do**
12: **if** $F^{\pm} := isedge(P^{'}, v^{+}, v^{-}) \neq \emptyset$ **then**
13: Find $v^{'} := [v^{+}, v^{-}] \cap h$
14: **if** $v^{'} \notin V^{'}$ **then**
15: $V^{'} \leftarrow V^{'} \cup \{v^{'}\}$, $V_h \leftarrow V_h \cup \{v^{'}\}$ and $F_{v^{'}} = F^{\pm} \cup \{h\}$;
16: **else**
17: $F_{v^{'}} \leftarrow F_{v^{'}} \cup F^{\pm} \cup \{h\}$ and $V_h \leftarrow V_h \cup \{v^{'}\}$;
18: **end if**
19: **for all** $f \in F_{\pm}$ **do**
20: $V_f \leftarrow V_f \cup \{v^{'}\}$
21: **end for**
22: **end if**
23: **end for**
24: **end for**
25: **end if**
26: $V^{''} = V^{'} \setminus V^{-}$ and $F^{''} = \cup_{v \in V^{''}} F_v$;
27: $V_f \leftarrow V_f \setminus V^{-}$ for $f \in F^{''}$;
28: **return** $P^{''}$ with vertices $V^{''}$, faces $F^{''}$ and respective adjacency lists.

H and $P^{'}$. Note that by construction, for each vertex $v^{+} \in V^{+}$ and $v^{-} \in V^{-}$, the intersection of h and the line segment between v^{+} and v^{-}, namely $[v^{+}, v^{-}]$, is a singleton and it is on the boundary of polyhedron $P^{''} = P^{'} \cap H$. Clearly, not all these intersection points are vertices of the updated polyhedron $P^{''}$. Indeed, only the ones for which $[v^{+}, v^{-}]$ is an edge of polyhedron $P^{'}$ yield a vertex. Hence, for all couples $(v^{+}, v^{-}) \in V^{+} \times V^{-}$, we check if there is an edge of $P^{'}$ between the two, using *isedge* subroutine (line 12).

If there exists an edge between v^{+} and v^{-}, we compute the intersection point of

h and $[v^+, v^-]$, namely $v^{'}$ (line 13). As stated above, $v^{'}$ is a vertex of the updated polyhedron $P^{''}$. Moreover, as $v^{'}$ is on hyperplane h, $v^{'}$ is added as an adjacent vertex of h and symmetrically, h is added as an adjacent face of $v^{'}$. Recall that $isedge(P^{'}, v^+, v^-)$ returns the set of faces $F^{\pm}$ of $P^{'}$ for which both v^+ and v^- is adjacent to as long as there is an edge between them. That is, it returns the set of faces for which the edge between v^+ and v^- lays on. Since $v^{'}$ is also on the line segment between v^+ and v^-, these set of faces are also adjacent to $v^{'}$. Hence, $F^{\pm}$ is also added as adjacent faces of vertex $v^{'}$. Note that it is also possible to obtain an existing vertex by intersecting h and $[v^+, v^-]$. In order to avoid duplications, first $v^{'}$ is checked if it is included in $V^{'}$. Only if $v^{'}$ is a new vertex, it is added to the set of vertices $V^{'}$. See, lines 14-18 of Procedure 2 (Part 2).

As a final step, the vertices which are not included in H, namely $v \in V^-$, are eliminated from $V^{'}$ in order to obtain the vertices $V^{''}$ of the updated polyhedron $P^{''}$. Similarly, the faces which are adjacent to some vertex in $V^{''}$ form the set of faces $F^{''}$ of $P^{''}$ (line 26). F_v for $v \in V^{''}$ remain as they are, and V_f for $f \in F^{''}$ becomes $V_f \setminus V^-$ (line 27). Then, the updated polyhedron $P^{''}$ is returned (line 28).

4.2 A Vertex Enumeration Algorithm for Bounded and Unbounded Polyhedrons

The double description method is used in order to solve the on-line vertex enumeration problem where the initial polyhedron is bounded. In order to solve an off-line vertex enumeration problem, one method is to call *onlinevert* iteratively. Of course, the initialization is non-trivial as one needs to start with a bounded initial polyhedron.

Throughout this section, we assume that an H-representation of a polyhedron P, which can be bounded or unbounded, is given as

$$P = \{x \in \mathbb{R}^n \mid A^T x \geq b\} = \bigcap_{k=1}^{m} H_k,$$

where $H_k = \{x \in \mathbb{R}^n \mid a_k^T x \geq b_k\}$.

Note that even if the polyhedron is unbounded, that is $\operatorname{recc} P \neq \{0\}$, the vertice set V of the polyhedron is included in a sufficiently large bounded set as there are only finitely many vertices. We assume that this initial bounded set, which contains the set of all vertices of P, is known. Indeed, we take a 'diamond shape' bounded region in $\mathbb{R}^n$. In order to generate initial polyhedron P^0 we follow the steps described below.

Initialization:

Step 1. We take $2n$ vertices (V^0) to be the positive and negative standard unit vectors multiplied by a large number M, that is $v_i = Me_i$ and $v_{n+i} = -Me_i$ where e_i is the unit vector with i^{th} component being 1.

Step 2. We consider the set T of all possible n-tuples consisting of -1 and 1's, that is, $t \in T$ is $t = (t_1, ..., t_n)$ where $t_i \in \{-1, 1\}$ for all $i = 1, \ldots, n$. Clearly, there are 2^n elements in T and each $t \in T$ corresponds to an orthant given by $\operatorname{cone}\{t_1 e_1, \ldots, t_n e_n\}$ in $\mathbb{R}^n$. We assume that there is exactly one face in each orthant. In particular, we consider 2^n faces (F^0) given by $f_t := \operatorname{conv}\{Mt_1 e_1, \ldots, Mt_n e_n\}$ for $t \in T$.

Step 3. We form the adjacency lists. Clearly, if $v \in f_t$, then $v \in V_{f_t}$ and $f_t \in F_v$.

Then, the algorithm iterates, by calling *onlinevert* iteratively. Algorithm 1, namely *vertenum*, takes $A \in \mathbb{R}^{n \times m}, b \in \mathbb{R}^m$ as its input, and returns a V-representation to P, that is, it returns the set of vertices V and the set of extreme directions D satisfying $P = \operatorname{conv}\{V\} + \operatorname{cone}\{D\}$.

As seen in the pseudo-code of Algorithm 1, it starts with the initial polyhedron P^0 (line 1) and at iteration k, using *onlinevert* function it intersects the current polyhedron P^{k-1} with the k^{th} halfspace H_k in order to obtain P^k (lines 3-6). After finding P^m, one needs to check if there is any vertex which lays on the boundary of initial polyhedron P^0. In case there is no such vertex, then P is

bounded and we have $P = P^m$. On the other hand, any vertex of P^m which is on the boundary of P^0 is not a vertex of polyhedron P; hence, it is excluded from the list of the vertices (line 9). Clearly, a vertex on $\mathrm{bd}\, P^0$ demonstrates that P is not bounded. Indeed, any edge of P^m that is formed by a vertex on $\mathrm{bd}\, P^0$ and a vertex that is not on $\mathrm{bd}\, P^0$ yields an extreme direction of P. Hence, in order to find the extreme directions of P, *isedge* function is called for all $v \in V^m \cap \mathrm{bd}\, P^0$ and $v^{'} \in V^m \setminus \mathrm{bd}\, P^0$. If there exists an edge between them, $d := \frac{v^{'}-v}{\|v^{'}-v\|}$ is an extreme direction of the recession cone of P and it is added to set D (lines 10-14). The algorithm returns $V = V^m$ and D (lines 17-18).

Algorithm 1 *vertenum*(A, b)

1: Initialize P^0 as described in Steps 1-3 (with vertices V^0 and faces F^0);
2: Initialize $D = \emptyset$ (the set of extreme directions);
3: **for** $k = 1 : m$ **do**
4: $\quad H_k = \{x \in \mathbb{R}^n \mid a_k^T x \geq b_k\}$;
5: $\quad P^k = onlinevert(P^{k-1}, H_k)$ (with vertices V^k and faces F^k);
6: **end for**
7: **for all** $v \in V^m$ **do**
8: $\quad$ **if** $v \in \mathrm{bd}\, P^0$ **then**
9: $\quad\quad V^m \leftarrow V^m \setminus \{v\}$;
10: $\quad\quad$ **for all** $v^{'} \in V^m \setminus \mathrm{bd}\, P^0$ **do**
11: $\quad\quad\quad$ **if** $isedge(v, v^{'}, P^m) \neq \emptyset$ **then**
12: $\quad\quad\quad\quad D \leftarrow D \cup \frac{v^{'}-v}{\|v^{'}-v\|}$;
13: $\quad\quad\quad$ **end if**
14: $\quad\quad$ **end for**
15: $\quad$ **end if**
16: **end for**
17: $V = V^m$;
18: **return** P with vertices V and extreme directions D.

Remark 4.2.1. *Note that for some applications of the vertex enumeration problem, the main task might be to find only the vertices of the polyhedron but not necessarily the extreme directions of its recession cone. For such applications it is possible not to perform lines 10-14 of Algorithm 1 and to return V only.*

4.3 Vertex Enumeration Algorithms for Unbounded Polyhedrons with Given Recession Cones

In this section, we discuss modifications of Algorithm 1 which work more efficiently whenever polyhedron P is unbounded and its recession cone is the positive orthant, $\mathbb{R}^n_+$. The motivation behind is that in Benson-type MOP algorithms the upper image is known to be unbounded and its recession cone is the ordering cone of the multiobjective optimization problem, namely, $\mathbb{R}^n_+$, see Remark 3.1.1.

Recall that Benson's algorithm starts by finding the 'ideal point' v^0; $\mathcal{P}^0 := \{v^0\} + \mathbb{R}^n_+$ is the initial outer approximation of the upper image $\mathcal{P}$; and in each iteration, the current outer approximation is updated by intersecting it with a halfspace. For the algorithms described here we assume that polyhedron $\mathcal{P}^0$ is given.

The first algorithm to be described is almost the same as Algorithm 1. The only difference is the initialization part. For the second one, we modify *onlinevert* function to be called within the algorithm.

4.3.1 Algorithm 2

We explain the initialization of the algorithm and the rest follows almost the same as in Algorithm 1.

Note that as there are finitely many vertices of $\mathcal{P}$, there exists a bounded polyhedron which contains the set of all vertices $\mathcal{V}$ of $\mathcal{P}$. Moreover, $\mathcal{V} \subseteq \{v^0\}+\mathbb{R}^n_+$. Then, it is known that there exists a sufficiently large number $M > 0$ such that $\mathcal{V} \subseteq P^0 := \{v^0\} + \operatorname{conv}\{0, Me_1, \ldots, Me_n\}$, where e_i are the unit vectors with i^{th} component being 1.

Initialization: Clearly, the initial polyhedron has $n + 1$ vertices, namely,

$V^0 = \{v^0, v^1, \ldots, v^n\}$, where $v^i = v^0 + Me_i$ for $i = 1, \ldots, n$. Moreover, the convex hull of any n vertices forms a face of it. Then, any n-combination of these $n+1$ vertices correspond to a face, that is, there are $\binom{n+1}{n} = n+1$ faces. More specifically, $F^0 = \{f^0, f^1, \ldots, f^n\}$ where $f^i = \operatorname{conv}\{V^0 \setminus \{v^i\}\}$. Clearly, the adjacent vertices of face f^i is $V_{f^i} = V^0 \setminus \{v^i\}$ and the adjacent faces of vertex i is $F_{v^i} = F^0 \setminus \{f^i\}$.

Lines 1-6 of Algorithm 1 is the same for Algortihm 2, where P^0 is formed as described above. After the main for loop, Algorithm 2 has a slight modification. Recall that for Algorithm 1, the vertices of polyhedron P^m which are on the boundary of initial polyhedron P^0 are 'artificial', that is, they are not vertices of polyhedron P and using these artificial vertices it is possible to construct the extreme directions of $\operatorname{recc} P$.

For Algorithm 2, only the vertices on face f^0 are 'artificial' by the construction of the initial polyhedron. Then, only the vertices on face f^0 of P^0 are eliminated from the set of vertices of the current (last) polyhedron. Hence, on line 8 of Algorithm 2 we check if $v \in f^0$ instead of checking $v \in \operatorname{bd} P^0$. Moreover, as the extreme directions are already known, one does not need to compute them as it is done in Algorithm 1.

Algorithm 2 *vertenum2*(A, b)

1: Initialize P^0 as described (with vertices V^0 and faces F^0);
2: Initialize $D = \emptyset$ (the set of extreme directions);
3: **for** $k = 1 : m$ **do**
4: $\quad H_k = \{x \in \mathbb{R}^n \mid a_k^T x \geq b_k\}$;
5: $\quad P^k = onlinevert(P^{k-1}, H_k)$ (with vertices V^k and faces F^k);
6: **end for**
7: **for all** $v \in V^m$ **do**
8: $\quad$ **if** $v \in f^0$ **then**
9: $\quad\quad V^m \leftarrow V^m \setminus \{v\}$;
10: $\quad$ **end if**
11: **end for**
12: $V = V^m$;
13: **return** P with vertices V and extreme directions D.

4.3.2 Algorithm 3

For this algorithm, it is also known that $\operatorname{recc} P = \mathbb{R}^n_+$, that is, the extreme directions are $D = \{e_1, \ldots, e_n\}$. The main difference of this algorithm from the others is that instead of starting with a sufficiently large bounded polyhedron, we use the extreme directions directly in computing the updated polyhedrons.

For the algorithm, we define 'adjacent faces of extreme directions' and 'adjacent directions of faces' as below. Note that the notation $v + \mathbb{R}_+ d$ stands for the ray emanating from vertex v in direction d, $\{v + kd \mid k \geq 0\}$.

Definition 4.3.1. *Let P be an unbounded polyhedron. An extreme direction $d \neq 0$ of* $\operatorname{recc} P$ *is said to be adjacent to a face f of P if there exists a vertex v of P such that $v + \mathbb{R}_+ d$ forms an edge of P and is on face f. Symmetrically, extreme direction d is said to be an* adjacent direction *of face f.*

Remark 4.3.2. *As before, F_v denotes the set of all adjacent faces of vertex v. Different from the previous ones, for this algorithm, V_f denotes the set of adjacent vertices together with the set of adjacent directions of face f. Moreover, F_d is the set of adjacent faces of extreme direction d. In a way, we treat the extreme directions as vertices. Hence, from now on whenever we mention adjacent vertices of a face, we mean the union of adjacent vertices and adjacent directions of it.*

For this algorithm, we modify *onlinevert* function such that it utilizes these modified adjacency structures. The structure of the main algorithm is then similar to the previous ones. The initialization and the pseudo-code for Algorithm 3 are as follows.

Initialization: Note that initial vertex v^0 is given and $P^0 = v^0 + \operatorname{cone} D$ is the initial polyhedron, where $D = \{e_1, \ldots, e_n\}$. Then, the set of vertices of the initial polyhedron is $V^0 = \{v^0\}$. Moreover, there are n faces given by $f_i = v^0 + \operatorname{cone}(D \setminus e_i)$. Hence $F^0 = \{f^1, \ldots, f^n\}$. The adjacency lists are initialized as follows: $F_{v^0} = \{f^1, \ldots, f^n\}$, $F_{e_i} = F \setminus \{f^i\}$ and $V_{f^i} = \{v^0\} \cup D \setminus \{e_i\}$ for $i = 1, \ldots, n$.

Algorithm 3 *vertenum*3(A, b)

1: Initialize P^0 as described (with vertices V^0, faces F^0 and directions D);
2: **for** $k = 1 : m$ **do**
3: $\quad H_k = \{x \in \mathbb{R}^n \mid a_k^T x \geq b_k\}$;
4: $\quad P^k = onlinevert2(P^{k-1}, H_k)$ (with vertices V^k, faces F^k and directions D);
5: **end for**
6: $V = V^m$;
7: **return** P with vertices V and extreme directions D.

As partially explained in Remark 4.3.2, for Algorithm 3, we treat the extreme directions (almost) as vertices. Note that *isedge* function, as given by Procedure 1, for given two vertices of polyhedron $P^{'}$ does not use the coordinates of the vertices but only the adjacency lists. Then, by definition of an adjacent face of an extreme direction d and by the usage of notation V_f as the union of adjacent vertices and adjacent directions of face f, $isedge(P^{'}, d, v)$ returns the set of faces on which $v + \mathbb{R}_+ d$ lays if this is an edge of $P^{'}$; and returns empty set if $v + \mathbb{R}_+ d$ is not an edge of it.

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

Whenever h is parallel to a direction d, it does not intersect any edge of the form $v + \mathbb{R}_+ d$. Instead, as long as it is not redundant, it cuts polyhedron $P^{'}$ in parallel to direction d. Hence, h is an adjacent face of direction d. Indeed, there must be a vertex v of $P^{''}$ such that $v + \mathbb{R}_+ d$ is an edge of $P^{''}$ and lays on h. Similarly, d is an adjacent direction of face h.

Now, let us describe the modified *onlinevert* function, namely, *onlinevert*2. Note that *onlinevert*2 takes polyhedron $P^{'}$ (with its vertices $V^{'}$, faces $F^{'}$, directions D and adjacency lists) and halfspace $H = \{x \in \mathbb{R}^n \mid a^T x \geq b\}$ as its inputs. It returns $P^{''} := P^{'} \cap H$ (with its vertices $V^{''}$, faces $F^{''}$, directions D and adjacency lists).

The general structure of the procedure is similar to *onlinevert*. It starts with creating the three subsets V^+, V^0, V^- of vertices of $P^{'}$ according to their positions with respect to H. See lines 1-10 of Procedure 3. Then, as in Procedure 2, it checks if all the vertices are in H; and it returns $P^{''} = P^{'}$ if this is the case (lines 11-12). Note that as P is unbounded, we cannot conclude that $P^{''} = \emptyset$ if $V^- = V^{'}$ as it is done in Algorithms 1 and 2.

Procedure 3 *onlinevert*2$(P^{'}, H)$

1: Initialize $V^0, V^+, V^- := \emptyset$;
2: **for all** $v \in V^{'}$ **do**
3: **if** $a^T v > b$ (i.e. $v \in \operatorname{int} H$) **then**
4: $V^+ \leftarrow V^+ \cup \{v\}$;
5: **else if** $a^T v = b$ (i.e. $v \in h$) **then**
6: $V^0 \leftarrow V^0 \cup \{v\}$;
7: **else if** $a^T v < b$ (i.e. $v \notin H$) **then**
8: $V^- \leftarrow V^- \cup \{v\}$;
9: **end if**
10: **end for**
11: **if** $V^+ \cup V^0 = V^{'}$ **then**
12: **return** $P^{''} = P^{'}$;
13: **else**
14: Do **Procedure 3 (Part 2)**;
15: **end if**
16: $V^{''} = V^{'} \setminus V^-$ and $F^{''} = \cup_{v \in V^{''}} F_v$;
17: $V_f \leftarrow V_f \setminus V^-$ for $f \in F^{''}$;
18: **return** $P^{''}$ with vertices $V^{''}$, faces $F^{''}$ and respective adjacency lists.

In case there exists at least one vertex out of H, the algorithm continues with Procedure 3 (Part 2). First, h is added as a new face and its adjacent vertices are introduced as empty set (line 1). As long as there are vertices that are already on the boundary of halfspace H, h is added as an adjacent face of these vertices and the vertices are added as adjacent vertices of h (line 3).

Then, the main loop goes over all vertices that are in int H and over all directions D (line 5). If v^+ is one of the directions, say $d \in D$, and if d is parallel to h, then d is added as an 'adjacent vertex' of h and h is added as an adjacent face of d (lines 6-7). Otherwise, i.e. when $v^+ \in V^+$ or when v^+ is a direction which is not parallel to h; the algorithm goes over all vertices v^- that are not included in

H and checks if there is an edge formed by v^+ and v^- using *isedge* function (line 10). If there exists an edge and if v^+ is a vertex, then vertex $v^{'}$ of the updated polyhedron is found as it was done also in Algorithms 1 and 2 (line 12). If v^+ is a direction and together with v^- it forms an edge then, it is known that edge given by $v^- + \mathbb{R}_+ v^+$ intersects h at a single point, say $v^{'}$ (line 14). In both cases, $v^{'}$ is a vertex of the updated polyhedron.

Procedure 3 (Part 2) *onlinevert2*$(P^{'}, H)$

```
1:  F' ← F' ∪ {h}, V_h = ∅;
2:  for all v ∈ V^0 do
3:     V_h ← V_h ∪ {v} and F_v ← F_v ∪ {h};
4:  end for
5:  for all v^+ ∈ V^+ ∪ D do
6:     if v^+ ∈ D and a^T v^+ = 0 (h is parallel to v^+) then
7:        V_h ← V_h ∪ {d}, F_d = F_d ∪ {h};
8:     else
9:        for all v^- ∈ V^- do
10:          if F^± := isedge(P', v^+, v^-) ≠ ∅ then
11:             if v^+ ∈ V^+ then
12:                Find v' := [v^+, v^-] ∩ h;
13:             else
14:                Find v' := (v^- + R_+ v^+) ∩ h;
15:             end if
16:             if v' ∉ V' then
17:                V' ← V' ∪ {v'}, V_h ← V_h ∪ {v'} and F_v' = F^± ∪ {h};
18:             else
19:                F_v' ← F_v' ∪ F^± ∪ {h} and V_h ← V_h ∪ {v'};
20:             end if
21:             for all f ∈ F_± do
22:                V_f ← V_f ∪ {v'};
23:             end for
24:          end if
25:       end for
26:    end if
27: end for
```

If $v^{'}$ is a vertex which is not already in $V^{'}$, then it is added to $V^{'}$ and to the adjacent vertices V_h of h. Moreover, the set of faces $F^{\pm}$ containing the edge that is formed by v^+ and v^-, as well as the new face h are added as adjacent faces of vertex $v^{'}$ (line 17). If $v^{'}$ is an already existing vertex, similar updates of the

adjacency lists are performed (line 19). Note that as $v^{'}$ is on faces in $F^{\pm}$, it is added as an adjacent vertex of each of them as well (lines 21-23).

4.4 An Illustrative Example

In order to illustrate Algorithms 1, 2 and 3, we solve a simple example using each of them separately. Since the second and the third algorithms assume that the recession cone of the polyhedron is the positive orthant, P is chosen accordingly. Moreover, we assume that v^0 which satisfies $v^0 + \mathbb{R}^n_+ \supseteq P$ is known.

For the first algorithm, v^0 is not taken as an input; while for the others we initialize using v^0. Consider the following simple example.

Example 4.4.1. *Let $n = 2$ and polyhedron P is given by $P = \{x \in \mathbb{R}^2 \mid A^T x \geq b\}$, where*

$$A = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad \textit{and} \quad b = \begin{bmatrix} -1 \\ -1 \\ 0 \\ -0.5 \end{bmatrix}$$

For the second and the third algorithms we assume that $v^0 = [-1,\ -1]^T$ is given and one needs to intersect initial polyhedron $P^0 := v^0 + \mathbb{R}^2_+$ with $\{x \in \mathbb{R}^2 \mid a_3^T x \geq b_3,\ a_4^T x \geq b_4\}$, where a_i is the i^{th} column of A and b_i is the i^{th} component of b.

4.4.1 Solution Using Algorithm 1

As $n = 2$, there are 4 vertices of the initial polyhedron P^0. Let's take $M = 5$ for illustrative purposes. Then, $V^0 = \{v^1, v^2, v^3, v^4\}$ where $v^1 = [5,\ 0]^T$, $v^2 = [0,\ 5]^T$, $v^3 = [-5,\ 0]^T$, $v^4 = [0,\ -5]^T$.

Similarly, there are four faces. Note that we have

$$T = \{(1,\ 1), (-1,\ 1), (1,\ -1), (-1,\ -1)\}$$

and each element of T defines a face. Hence $F^0 = \{f^1, f^2, f^3, f^4\}$ where

$$f^1 = \text{conv}\,\{[5,\ 0]^T, [0,\ 5]^T\},\ \ f^2 = \text{conv}\,\{[-5,\ 0]^T, [0,\ 5]^T\},$$

$$f^3 = \text{conv}\,\{[5,\ 0]^T, [0,\ -5]^T\},\ \ f^4 = \text{conv}\,\{[-5,\ 0]^T, [0,\ -5]^T\}.$$

The adjacency lists for P^0 is clearly $F_{v^1} = \{f^1, f^3\}, F_{v^2} = \{f^1, f^2\}, F_{v^3} = \{f^2, f^4\}$ and $F_{v^4} = \{f^3, f^4\}$. Similarly, $V_{f^1} = \{v^1, v^2\}, V_{f^2} = \{v^2, v^3\}, V_{f^3} = \{v^1, v^4\}$ and V_{f^4}=$\{v^3, v^4\}$, see Figure 4.1.

At iteration 1 of Algorithm 1 P^0 is intersected with $H_1 = \{x\ |x_1 \geq -1\}$, see Figure 4.2. P^1 is computed by calling $onlinevert(P^0, H_1)$. Note that, we obtain $V^- = \{v^3\}$,$V^0 = \emptyset$, $V^+ = \{v^1, v^2, v^4\}$, see Procedure 2. As there are vertices both in V^- and V^+, $h^1 \cap P^0 = \{x\ |x \in P^0, x_1 = -1\}$ is a new face, namely f^5 (Procedure 2 Part 2 line 6).

For $v^+ = v^1$, $v^- = v^3$, $F^{\pm} := isedge(P^0, v^+, v^-) = \emptyset$. As there is no edge between v^1 and v^3; the algorithm continues by checking $v^+ = v^2$, $v^- = v^3$. Then, $F^{\pm} := isedge(P^0, v^+, v^-) = \{f^2\}$. As vertex $v^{'} := [v^+, v^-] \cap h^1 = [-1,\ 4]^T \notin V^0$, $v^5 = v^{'}$ is a new vertex and added to V^0. Moreover, V_{f^5}=$\{v^5\}$, F_{v^5}=$\{f^2, f^5\}$ (line 15), and $V_{f^2} = \{v^2, v^3, v^5\}$ (line 20).

Similarly, for $v^+ = v^4$, $v^- = v^3$, $isedge(P^0, v^+, v^-) = \{f^4\}$. Then, since $v^{'} := [v^+, v^-] \cap h^1 = [-1,\ -4]^T \notin V^0$, we add $v^6 := v^{'}$ to V^0. Moreover, V_{f^5}=$\{v^5, v^6\}$, F_{v^6}=$\{f^4, f^5\}$ (line 15), and V_{f^4}=$\{v^3, v^4, v^6\}$ (line 20).

At the end, we have $V^1 = \{v^1, v^2, v^4, v^5, v^6\}$, $F_{v^1} = \{f^1, f^3\}$, $F_{v^2} = \{f^1, f^2\}$,$F_{v^4} = \{f^3, f^4\}$, $F_{v^5} = \{f^2, f^5\}$, $F_{v^6} = \{f^4, f^5\}$. Hence, $F^1 = \{f^1, f^2, f^3, f^4, f^5\}$ (line 26). Moreover, V_{f^2}=$\{v^2, v^5\}$, V_{f^4}=$\{v^4, v^6\}$, V_{f^5}=$\{v^5, v^6\}$ and V_{f^i} remains the same for $i = 1, 3$ (line 27).

At iteration 2, P^1 is intersected with $H_2 = \{x\ |x_2 \geq -1\}$, see Figure 4.3. P^2 is computed by calling $onlinevert(P^1, H_2)$. Clearly, $V^- = \{v^4, v^6\}$,$V^0 = \emptyset$,

$V^+ = \{v^1, v^2, v^5\}$ and $h^2 \cap P^1 = \{x \mid x \in P^1, x_2 = -1\}$ is a new face, namely f^6. (Procedure 2 Part 2 line 6).

For $v^+ = v^1$, $v^- = v^4$, $F^\pm := isedge(P^1, v^+, v^-) = \{f^3\}$. As vertex $v' := [v^+, v^-] \cap h^2 = [4, -1]^T \notin V^1$, $v^7 = v'$ is a new vertex and added to V^1. Moreover, V_{f^6}=$\{v^7\}$; F_{v^7}=$\{f^3, f^6\}$ (line 15), and $V_{f^3} = \{v^1, v^4, v^7\}$ (line 20). For $v^+ = v^1$, $v^- = v^6$, $F^\pm := isedge(P^1, v^+, v^-) = \emptyset$, that is, there is no edge between v^1 and v^6.

The algorithm continues by checking $v^+ = v^2$, $v^- = v^4$ and $v^+ = v^2$, $v^- = v^6$. For both cases there are no edges in between v^+ and v^-.

Similarly, there is no edge between $v^+ = v^5$, $v^- = v^4$. Then we check $v^+ = v^5$, $v^- = v^6$ and find $F^\pm := isedge(P^1, v^+, v^-) = \{f^5\}$. As vertex $v' := [v^+, v^-] \cap h^2 = [-1, -1]^T \notin V^1$, $v^8 = v'$ is a new vertex and added to V^1. Moreover, V_{f^6}=$\{v^7, v^8\}$; F_{v^8}=$\{f^5, f^6\}$ (line 15), and $V_{f^5} = \{v^5, v^6, v^8\}$ (line 20).

At the end of second iteration, we have $V^2 = \{v^1, v^2, v^5, v^7, v^8\}$, $F_{v^1} = \{f^1, f^3\}$, $F_{v^2} = \{f^1, f^2\}$,$F_{v^5} = \{f^2, f^5\}$, $F_{v^7} = \{f^3, f^6\}$, $F_{v^8} = \{f^5, f^6\}$. Hence, $F^2 = \{f^1, f^2, f^3, f^5, f^6\}$ (line 26). Moreover, V_{f^3}=$\{v^1, v^7\}$, V_{f^5}=$\{v^5, v^8\}$, V_{f^6}=$\{v^7, v^8\}$ and V_{f^i} remains the same for $i = 1, 2$ (line 27).

At iteration 3, we add half-space $H^3 = \{x \mid x_1 + x_2 \geq 0\}$ where the corresponding hyperplane is $h^3 = \{x \mid x_1 + x_2 = 0\}$. See Figure 4.4. We obtain $V^- = \{v^8\}$, $V^0 = \emptyset$, $V^+ = \{v^1, v^2, v^5, v^7\}$ and hence $h^3 \cap P^2$ is a new face, namely f^7.

There are no edges between, v^1 and v^8 as well as v^2 and v^8. For $v^+ = v^5$, $v^- = v^8$, $isedge(P^2, v^+, v^-) = \{f^5\}$. Then, since $v' := [v^+, v^-] \cap h^3 = [-1, 1]^T \notin V^2$ we add $v^9 = v'$ to the set of vertices. Moreover, V_{f^7}=$\{v^9\}$; F_{v^9}=$\{f^5, f^7\}$ (line 15), and $V_{f^5} = \{v^5, v^8, v^9\}$ (line 20). Similarly, for $v^+ = v^7$, $v^- = v^8$, $isedge(P^2, v^+, v^-) = \{f^6\}$. Then, $v' := [v^+, v^-] \cap h^3 = [1, -1]^T \notin V^2$ we add $v^{10} = v'$ to the set of vertices; V_{f^7}=$\{v^9, v^{10}\}$; $F_{v^{10}}$=$\{f^6, f^7\}$ (line 15), and $V_{f^6} = \{v^7, v^8, v^{10}\}$ (line 20).

At the end of third iteration, we have $V^3 = \{v^1, v^2, v^5, v^7, v^9, v^{10}\}$, $F_{v^1} =$

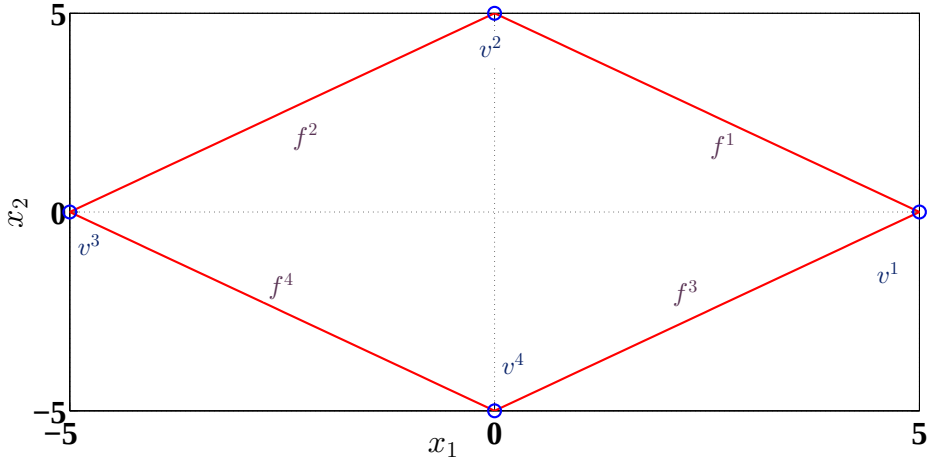

Figure 4.1: Initial polyhedron P^0 of Algorithm 1 for Example 4.4.1

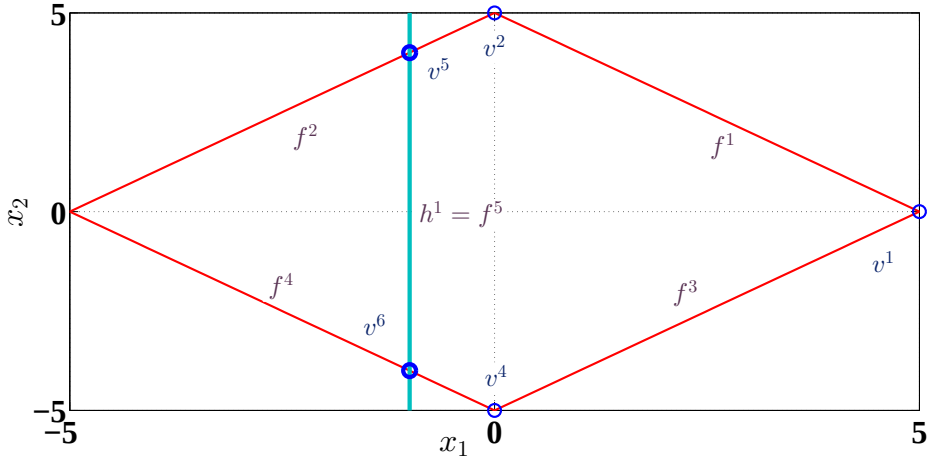

Figure 4.2: Iteration 1 of Algorithm 1 for Example 4.4.1

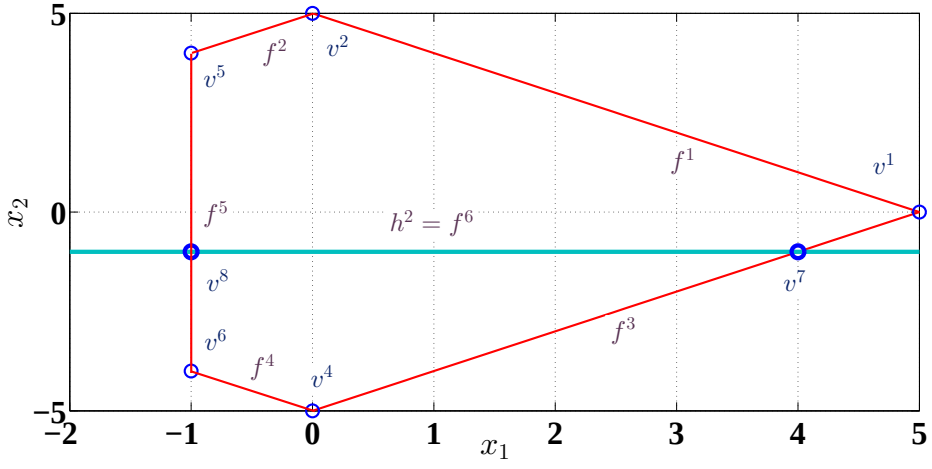

Figure 4.3: Iteration 2 of Algorithm 1 for Example 4.4.1

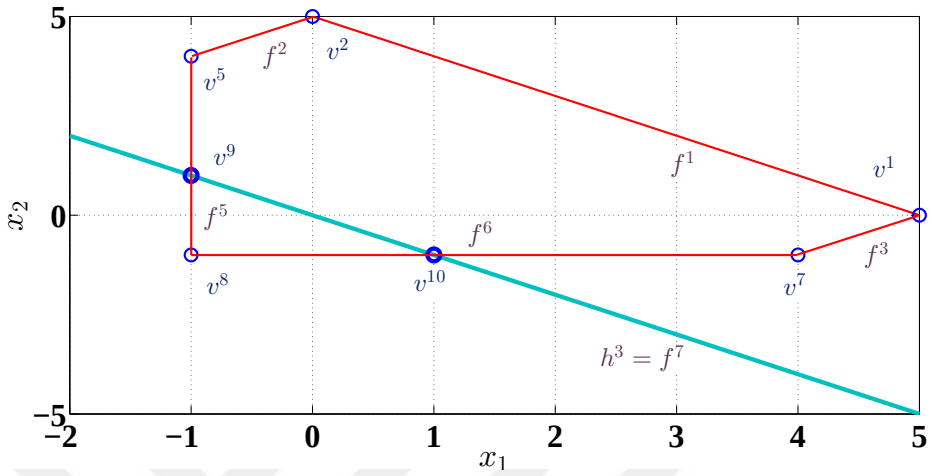

Figure 4.4: Iteration 3 of Algorithm 1 for Example 4.4.1

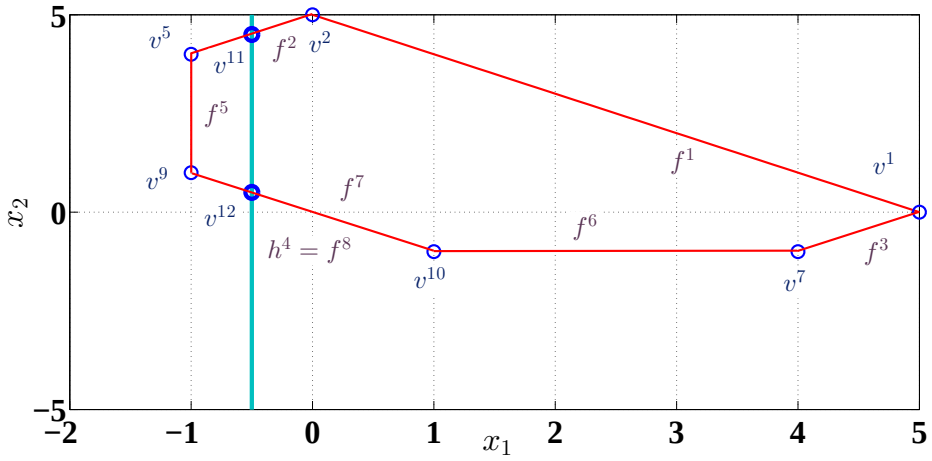

Figure 4.5: Iteration 4 of Algorithm 1 for Example 4.4.1

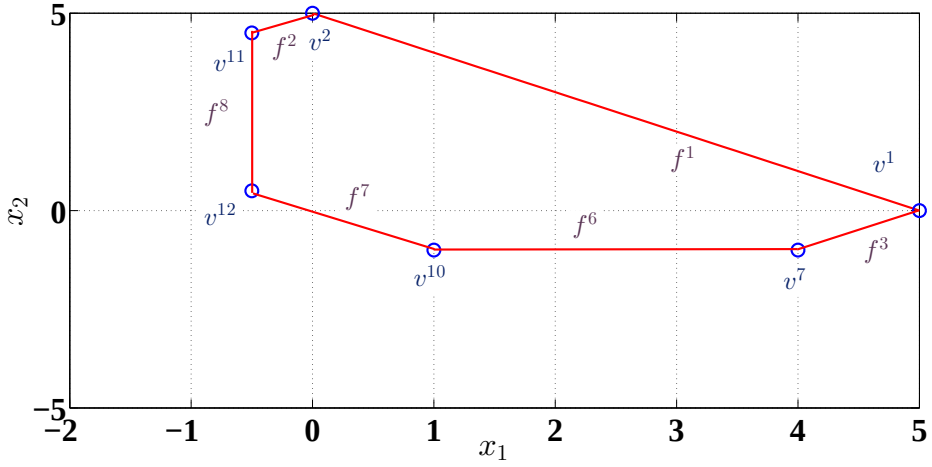

Figure 4.6: The end of iteration 4 of Algortithm 1 for Example 4.4.1

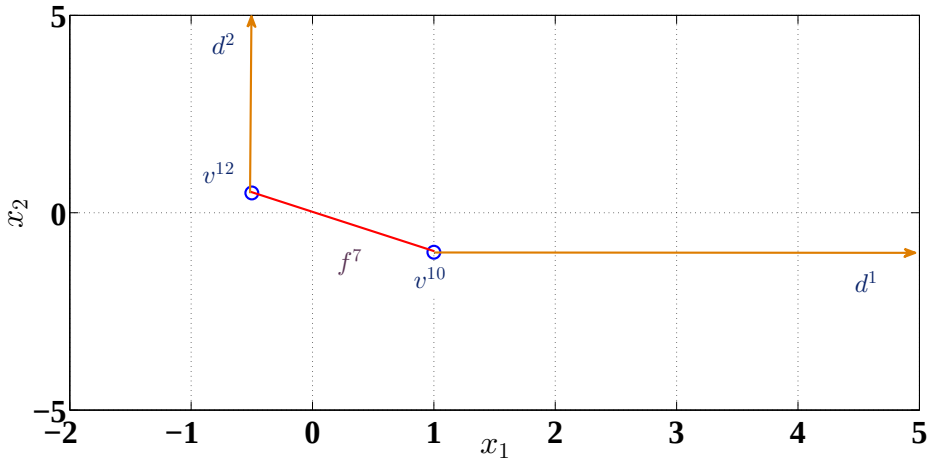

Figure 4.7: V-representation for P of Example 4.4.1

$\{f^1, f^3\}$, $F_{v^2} = \{f^1, f^2\}$,$F_{v^5} = \{f^2, f^5\}$, $F_{v^7} = \{f^3, f^6\}$, $F_{v^9} = \{f^5, f^7\}$, $F_{v^{10}} = \{f^6, f^7\}$. Hence, $F^3 = \{f^1, f^2, f^3, f^5, f^6, f^7\}$ (line 26). Moreover, V_{f^5}=$\{v^5, v^9\}$, V_{f^6}=$\{v^7, v^{10}\}$, V_{f^7}=$\{v^9, v^{10}\}$ and V_{f^i} remains the same for $i = 1, 2, 3$ (line 27).

At the last iteration P^3 is intersected with $H^4 = \{x \,|x_1 \geq -0.5\}$ and we obtain $V^- = \{v^5, v^9\}$,$V^0 = \emptyset$, $V^+ = \{v^1, v^2, v^7, v^{10}\}$, see Figure 4.5. Following the same steps as before we obtain, $V^4 = \{v^1, v^2, v^7, v^{10}, v^{11}, v^{12}\}$, where $v^{11} = [-0.5,\ 4.5]^T$ and $v^{12} = [-0.5,\ 0.5]^T$. $F_{v^1} = \{f^1, f^3\}$, $F_{v^2} = \{f^1, f^2\}$,$F_{v^7} = \{f^3, f^6\}$, $F_{v^{10}} = \{f^6, f^7\}$, $F_{v^{11}} = \{f^2, f^8\}$, $F_{v^{12}} = \{f^7, f^8\}$. Hence, $F^4 = \{f^1, f^2, f^3, f^6, f^7, f^8\}$ (line 26). Moreover, V_{f^1}=$\{v^1, v^2\}$, V_{f^2}=$\{v^2, v^{11}\}$, V_{f^3}=$\{v^1, v^7\}$ V_{f^6}=$\{v^7, v^{10}\}$, V_{f^7}=$\{v^{10}, v^{12}\}$, V_{f^8}=$\{v^{11}, v^{12}\}$ (line 27). See Figure 4.6.

Going back to lines 7-17 of Algorithm 1, we check if $v \in V^4$ is on the boundary of the initial polyhedron. Then, v^1, v^2, v^7 and v^{11} are not vertices of the unbounded polyhedron P (line 9). Moreover, as there is an edge between v^{10} and v^7, $d^1 := \frac{v^7 - v^{10}}{\|v^7 - v^{10}\|} = [1,\ 0]^T = e_1$ is an extreme direction of P. Similarly, $d^2 := \frac{v^{11} - v^{12}}{\|v^{11} - v^{12}\|} = [0,\ 1]^T = e_2$ is an extreme direction (line 12). Hence, $P = \text{conv}\,\{v^{10}, v^{12}\} + \text{cone}\,\{d^1, d^2\}$. See Figure 4.7.

Remark 4.4.2. *Note that at the end of iteration 2, the first vertex which is not on the boundary of P^0, namely $v^8 = [-1,\ -1]^T$, is found. If the extreme directions are known to be e_1 and e_2 then, it is clear that $v^8 + \text{cone}\,\{e_1, e_2\}$ contains*

polyhedron P.

This situation is not a coincidence. Indeed it is guaranteed by the structure of the example that we consider. Recall that we aim to use these vertex enumeration algorithms within Benson-type MOP algorithms, where the initial step is to find the ideal point. The ideal point can be found by intersecting n halfspaces whose normal directions are the unit vectors $e_i, i = 1, \ldots, n$. In this example, in order to illustrate the structure of Benson-type MOP algorithms, the first two halfspaces are written such that their normal directions are e_2 and e_1, respectively.

For Algorithms 2 and 3 we assume that $[-1,\ -1]^T$ is given as v^0. The initial polyhedrons are constructed accordingly and in order to compute the V-representation of P, the initial polyhedron is intersected with H^3 and H^4 only.

4.4.2 Solution Using Algorithm 2

As explained in Remark 4.4.2, the initial vertex $v^0 = [-1,\ -1]^T$ is given. As in the previous case, we take M value as 5. Then, there are three vertices of the initial polyhedron, namely $V^0 = \{v^0, v^1, v^2\}$, where $v^1 = v^0 + 5e_1 = [4,\ -1]^T$ and $v^2 = v^0 + 5e_2 = [-1,\ 4]^T$ and $P^0 = \text{conv}\,\{v^0, v^1, v^2\}$. Moreover, there are three faces, that is $F^0 = \{f^0, f^1, f^2\}$, where $f^0 := \text{conv}\,\{v^1, v^2\}$, $f^1 := \text{conv}\,\{v^0, v^2\}$ and $f^2 := \text{conv}\,\{v^0, v^1\}$. Clearly,

$$F_{v^0} = \{f^1, f^2\},\ F_{v^1} = \{f^0, f^2\},\ F_{v^2} = \{f^0, f^1\},$$

$$V_{f^0} = \{v^1, v^2\},\ V_{f^1} = \{v^0, v^2\},\ V_{f^2} = \{v^0, v^1\}.$$

See Figure 4.8.

At iteration 1, P^0 is intersected with half-space $H^3 = \{x\ |x_1 + x_2 \geq 0\}$, see Figure 4.9. Similar to Algorithm 1, $onlinevert(P^0, H^3)$ is called. As a result, $f^3 := h^3 \cap P^0$ is added to F^0 and $v^3 = [1,\ -1]^T$ and $v^4 = [-1,\ 1]^T$ are added to V^0. Since $V^- = \{v^0\}$, at the end of iteration 1 we have $V^1 = \{v^1, v^2, v^3, v^4\}$, $F_{v^1} =$

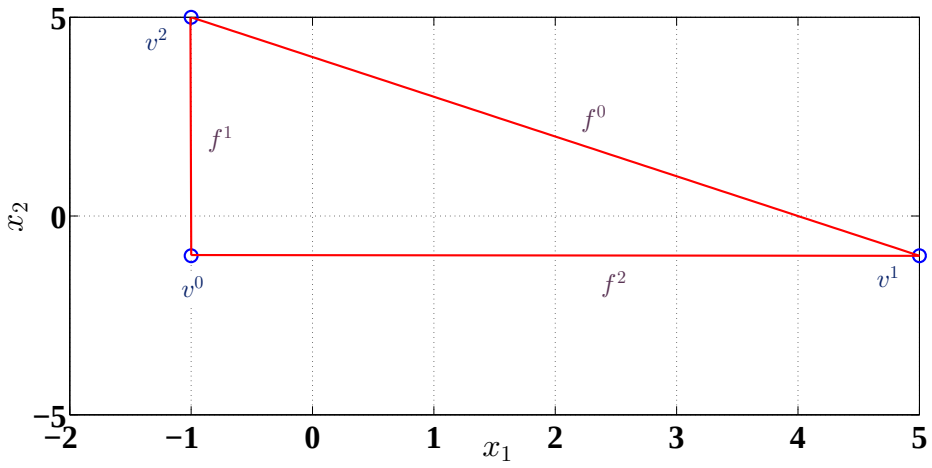

Figure 4.8: Initial polyhedron P^0 of Algorithm 2 for Example 4.4.1

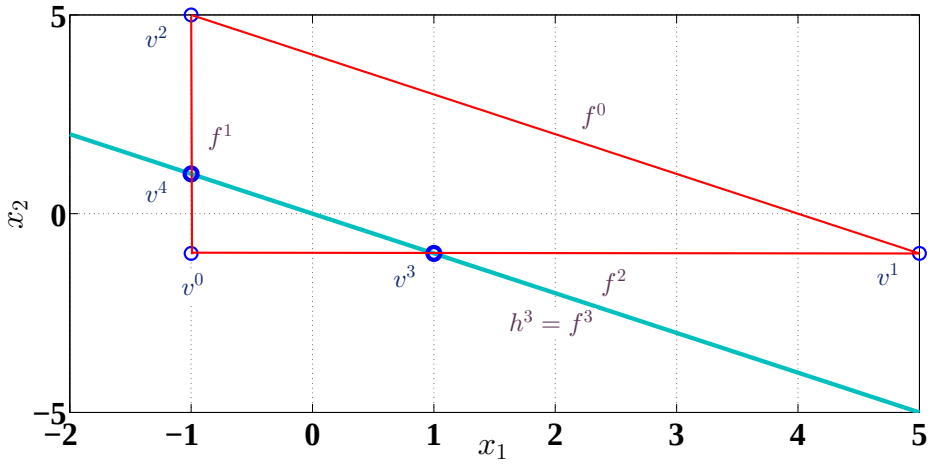

Figure 4.9: Iteration 1 of Algorithm 2 for Example 4.4.1

$\{f^0, f^2\}$, $F_{v^2} = \{f^0, f^1\}$, $F_{v^3} = \{f^2, f^3\}$, $F_{v^4} = \{f^1, f^3\}$ and $F^1 = \{f^0, f^1, f^2, f^3\}$, $V_{f^0} = \{v^1, v^2\}, V_{f^1} = \{v^2, v^4\}$, $V_{f^2} = \{v^1, v^3\}$, $V_{f^3} = \{v^3, v^4\}$.

At iteration 2, P^1 is intersected with half-space $H^4 = \{x \mid x_1 \geq -0.5\}$, see Figure 4.10. As a result of $onlinevert(P^1, H^4)$, $f^4 := h^4 \cap P^1$ is added to F^1 and $v^5 = [-0.5,\ 3.5]^T$ and $v^6 = [-0.5,\ 0.5]^T$ are added to V^1. Since $V^- = \{v^2, v^4\}$, at the end of iteration 2 we have $V^2 = \{v^1, v^3, v^5, v^6\}$, $F_{v^1} = \{f^0, f^2\}$, $F_{v^3} = \{f^2, f^3\}$, $F_{v^5} = \{f^0, f^4\}$, $F_{v^6} = \{f^3, f^4\}$ and $F^2 = \{f^0, f^2, f^3, f^4\}$, $V_{f^0} = \{v^1, v^5\}, V_{f^2} = \{v^1, v^3\}$, $V_{f^3} = \{v^3, v^6\}$, $V_{f^4} = \{v^5, v^6\}$.

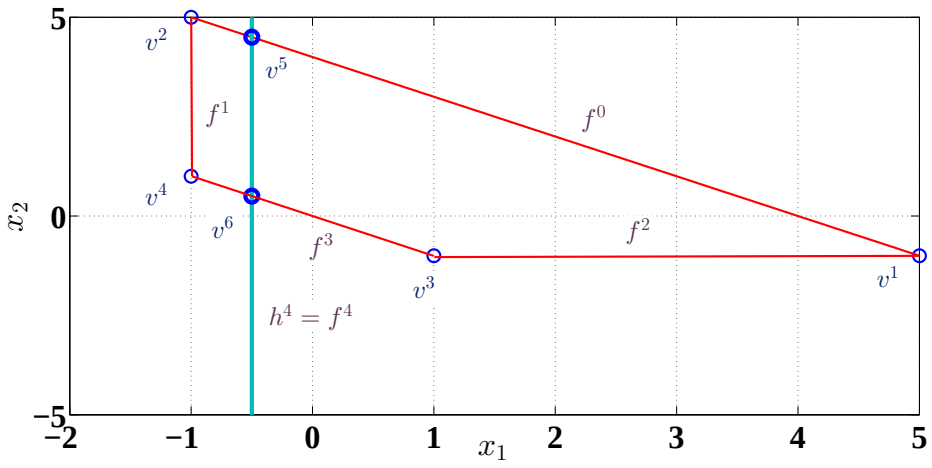

Figure 4.10: Iteration 2 of Algorithm 2 for Example 4.4.1

Then, we check if $v \in V^2$ is on f^0, see lines 7-11 of Algorithm 2. Accordingly, v^1 and v^5 are not vertices of the unbounded polyhedron P (line 9). Hence, the only vertices are v^3 and v^6. Since the extreme directions are known, the V-representation for P is given by $P = \text{conv}\,\{v^3, v^6\} + \text{cone}\,\{e_1, e_2\}$, as in Figure 4.7.

4.4.3 Solution Using Algorithm 3

The initial vertex v^0 is given as $[-1,\ 1]^T$ as explained in Remark 4.4.2. Moreover, it is given that $D = \{e_1, e_2\}$. Then, the initial polyhedron has a single vertex, that is $V^0 = \{v^0\}$, and it has two faces, that is $F^0 = \{f^1, f^2\}$ where $f^1 = v^0 + \text{cone}\,\{e_2\}$ and $f^2 = v^0 + \text{cone}\,\{e_1\}$. Clearly, $F_{v^0} = \{f^1, f^2\}, F_{e_1} = \{f^2\}, F_{e_2} = \{f^1\}$ and $V_{f^1} = \{v^0, e_2\}, V_{f^2} = \{v^0, e_1\}$. See Figure 4.11.

At iteration 1, P^0 is intersected with $H^3 = \{x \mid x_1 + x_2 \geq 0\}$, see Figure 4.12. Algorithm 3 calls $onlinevert2(P^0, H^3)$ and we obtain $V^- = \{v^0\}$, $V^0 = \emptyset$, and $V^+ = \emptyset$, see Procedure 3 lines 1-10. Since there are vertices out of H^3 (see lines 11-13), the corresponding hyperplane h^3 defines a new face, hence we have $F^0 = \{f^1, f^2, f^3\}$ where $f^3 = h^3 \cap P^0$, see Procedure 3 Part 2 line 1.

Then, we check each $v^+ \in V^+ \cup D$. Let $v^+ = e_1$. Note that $e_1 \in D$ but it is not

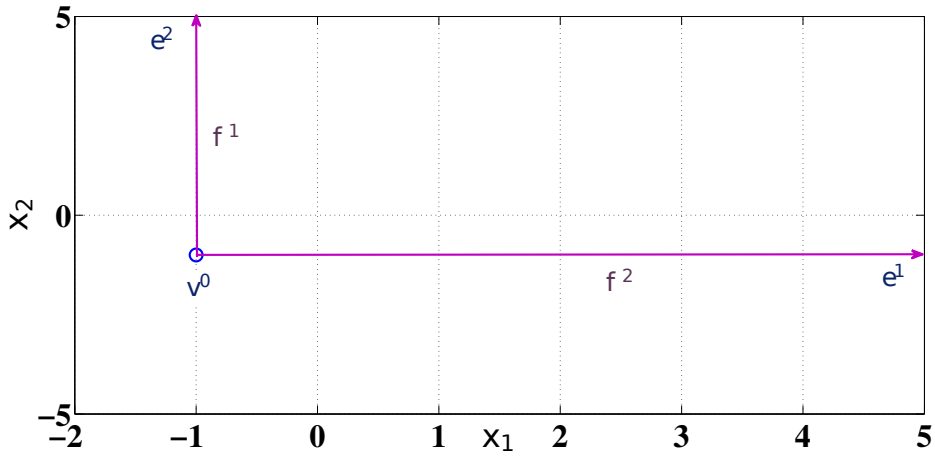

Figure 4.11: Initial polyhedron P^0 of Algorithm 3 for Example 4.4.1

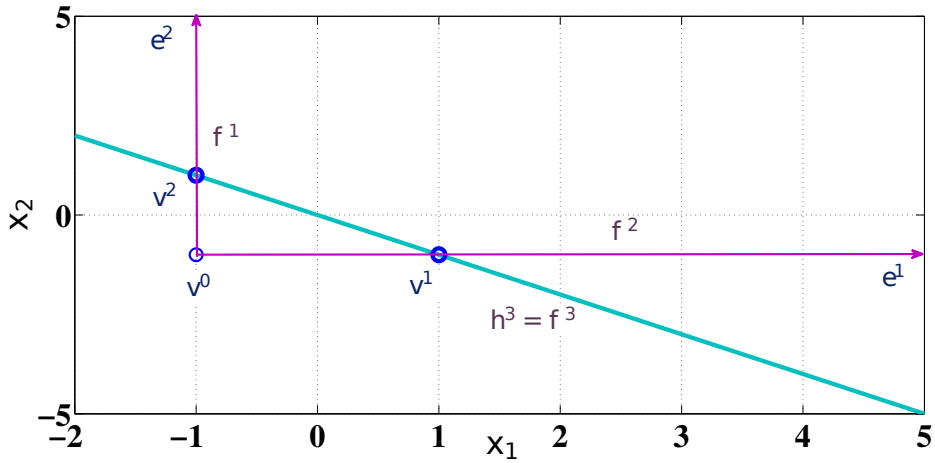

Figure 4.12: Iteration 1 of Algorithm 3 for Example 4.4.1

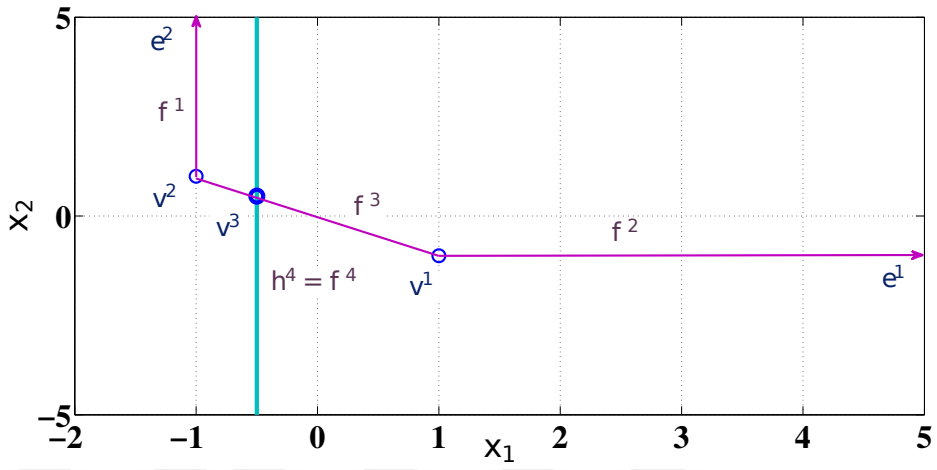

Figure 4.13: Iteration 2 of Algorithm 3 for Example 4.4.1

parallel to h^3, hence we consider $v^- = v^0$ and obtain $isedge(P^0, v^+, v^-) = \{f^2\}$, lines 5-10 of Procedure 3 Part 2. Then, we find $v^{'} := v^0 + \mathbb{R}_+ e_1 \cap h^3 = [0,\ -1]^T$, line 14. Since, $v^{'} \notin V^0$, we add $v^1 = v^{'}$ to the set of vertices. In particular, we have $V^0 = \{v^0, v^1\}, V_{f^3}$=$\{v^1\}$ and F_{v^1}=$\{f^2, f^3\}$ (line 17) and V_{f^2}=$\{v^0, e_1, v^1\}$ (lines 21-23).

Similarly, $v^+ := e_2$ is also not parallel to h^3 (line 6). Hence we let $v^- = v^0$ and obtain $isedge(P^0, v^+, v^-) = \{f^1\}$ (line 10). Then, since $v^{'} := (v^0 + \mathbb{R}_+ e_2) \cap h^3 = [0,\ 1]^T \notin V^0$ we add $v^2 = v^{'}$ to the set of vertices and obtain $V^0 = \{v^0, v^1, v^2\}$. Moreover, V_{f^3}=$\{v^1, v^2\}$, F_{v^2}=$\{f^1, f^3\}$ and V_{f^1}=$\{v^0, e_2, v^2\}$, see lines 17 and 22.

At the end of iteration 1, we obtain P^1 with vertices $V^1 = \{v^1, v^2\}$. Note that $F_{v^1} = \{f^2, f^3\}, F_{v^2} = \{f^1, f^3\}, F_{e_1} = \{f^2\}$ and $F_{e_2} = \{f^1\}$. Hence, the set of faces is $F^1 = \{f^1, f^2, f^3\}$, see line 16 of Procedure 3. Moreover, $V_{f^1} = \{e_2, v^2\}$ and V_{f^2}=$\{e_1, v^1\}$and $V_{f^3} = \{v^1, v^2\}$, line 17.

At iteration 2, P^1 is intersected with $H^4 = \{x\ |x_1 \geq -0.5\}$, see Figure 4.13. Clearly, $onlinevert2(P^1, H^4)$ is called, line 4 of Algorithm 3. Then, we obtain $V^- = \{v^2\}$,$V^0 = \emptyset$, $V^+ = \{v^1\}$, see lines 1-10 of Procedure 3. As there are vertices in V^-, the algorithm continues with Procedure 3 Part 2 line 1 and $F^1 = \{f^1, f^2, f^3, f^4\}$ where $f^4 = h^4 \cap P^1$.

Let $v^+ = v^1$. Note that $v_1 \notin D$. Hence we let $v^- = v^2$ (lines 5-9). Then, $isedge(P^1, v^+, v^-) = \{f^3\}$ is found (line 10). As $v^+ = v^1$ is a vertex but not a direction, we find $v^{'}$ as $[v^1, v^2] \cap h^4 = [-0.5,\ -0.5]^T$, line 12. Then, $v^3 := v^{'}$ is added as a new vertex, $V^1 = \{v^1, v^2, v^3\}$, and $V_{f^4} = \{v^3\}, F_{v^3} = \{f^3, f^4\}$ (line 17). Moreover, we have $V_{f^3} = \{v^1, v^2, v^3\}$.

The algorithm continues by checking $v^+ = e_1$. Note that this is a direction but not parallel to h^4. Hence, we let $v^- = v^2$ and find out that there is no edge starting from v^2 and goes through the direction e_1 since we have $isedge(v^+, v^-) = \emptyset$.

Then, we let $v^+ = e_2$. Note that e_2 is parallel to h^4 as we have $[1,\ 0]e_2 = 0$. Then, $V_{f^4} = \{v^3, e_2\}$ and $F_{e_2} = \{f^1, f^4\}$, see line 7 of Procedure 3 Part 2. Next, we update the vertices and the faces after the second iteration as in lines 16-17 of Procedure 3 (Part 1). Accordingly, $V^2 = \{v^1, v^3\}$ and we have $F_{v^1} = \{f^2, f^3\}, F_{v^3} = \{f^3, f^4\}, F_{e_1} = \{f^2\}$ and $F_{e_2} = \{f^4\}$. Hence, the set of faces is $F^2 = \{f^2, f^3, f^4\}$. Moreover, $V_{f^2} = \{e_1, v^1\}$ and V_{f^3}=$\{v^1, v^3\}$and $V_{f^4} = \{v^3, e_2\}$.

At the end of the second iteration, Algorithm 3 returns $V = V^2 = \{v^1, v^3\}$ and the extreme directions D. Hence, we obtain P as given in Figure 4.7.

Chapter 5

Computational Results

The algorithms that are described in Chapter 4 are implemented using MATLAB. Note that there is also a MATLAB implementation, namely 'c-bensolve', of a Benson-type convex multiobjective optimization algorithm [19]. As explained in Remark 3.1.1, an on-line vertex enumeration problem is solved in each iteration of c-bensolve. Indeed, c-bensolve generates outer approximations to the boundary of the upper image (or the set of weakly nondominated points in the objective space), by intersecting the current approximation with another halfspace in each iteration until the approximation is tight.

As a matter of fact, for a single multiobjective optimization problem, Benson's algorithm iteratively solves a single off-line vertex enumeration problem. The main point is that in each iteration of this algorithm halfspace H_k to be intersected with the current polyhedron is not given as an input but it is found by solving a scalarization problem for the corresponding multiobjective optimization problem.

In the current implementation of c-bensolve, these vertex enumeration problems are solved using a MATLAB function (*vert*) written for MATLAB implementation of Benson-type linear multiobjective optimization algorithm, namely 'bensolve' [46]. Even though an on-line vertex enumeration problem is solved at

each iteration of Benson-type algorithms, *vert* solves an off-line vertex enumeration problem. Hence, at each iteration, it computes all the vertices from an H-representation of the current outer approximation even though many vertices are already found in earlier iterations.

Instead of solving an off-line vertex enumeration problem at each iteration, we solve an on-line vertex enumeration problem by employing the MATLAB implementations of the procedures that are described in Chapter 4, within c-bensolve. In order to compare the performances of Algorithms 1-3, we randomly generate linear multiobjective optimization problems; we treat each problem as a vertex enumeration problem where H^k's are not direct inputs of the vertex enumeration problem but they are generated as they are done (by solving scalarizations) in Benson-type algorithms. We measure the CPU times that is spent only during *onlinevert* or *onlinevert*2 procedures. Note that both Algorithm 1 and 2 use *onlinevert* procedure but as the initializations of the two algorithms are different, the number of vertices of the current polyhedrons differs.

For the first algorithm, initial vertex v^0 is found by calling *onlinevert* n times, where n is the dimension (of the objective space). For the other algorithms, v^0 is given as the ideal point, see also Remark 4.4.2. In order to have a fair comparison, we treat the first n iterations of Algorithm 1 as part of the initialization step of Benson's algorithm. Hence, the $(n+1)^{st}$ iteration is counted as the first iteration. Note that the reason behind is also clear if one considers the illustrative example given in Section 4.4. When we solve Example 4.4.1 by Algorithms 2 and 3, we only consider H^3 and H^4, whereas when we solve it by Algorithm 1, we consider two more halfspaces, namely H^1 and H^2, in order to find v^0.

5.1 Creating Random Problems

In order to test the performances of the algorithms, we randomly generate linear multiobjective optimization problems in the following form:

$$\text{minimize } P^T x \text{ subject to } Ax \leq b, \;\; x \geq 0,$$

where $P \in \mathbb{R}^{n \times N}$ and $A \in \mathbb{R}^{m \times N}$ are matrices of respective sizes and $b \in \mathbb{R}^m$ is an m-dimensional vector. Clearly, the number of objectives is n, the number of variables is N and the number of constraints is m. As the objective space is n-dimensional, the vertex enumeration problem to be solved is also n-dimensional.

In order to generate random linear MOPs of this form, it is enough to generate matrices P and A as well as vector b, randomly. For our tests, each component of A and P is generated using independent normal distributions with mean $\mu = 0$ and variance $\sigma^2 = 100$, whereas each component of vector b is generated using independent uniform distributions on range $[0, 10]$. Note that we use a uniform distribution on this range in order to guarantee that we generate feasible linear programs. Moreover, in order to avoid numerical complications, we round each component of A, P and b to its closest integer.

When we generate a problem, we first check the feasibility and boundedness of it and add it to our sample only if the problem is solvable, that is, if it is bounded and feasible. Otherwise, we continue generating another set of P, A and b.

5.2 Results and Discussion

We consider different set of problems where the number of objectives (n) ranges from 2 to 5; the number of variables (N) ranges from 5 to 50; and the number of constraints (m) is taken as $2N$. For the problems with two objectives, for each choice of N and m, we generate 100 feasible and bounded linear MOPs. Similarly, for $n = 3$ and $n = 4$ we generate 50; and for $n = 5$, we generate 30 solvable problems for each choice of N, m.

These problems are solved using c-bensolve for a given error level ϵ, where ϵ is taken as 0.005 for biobjective problems and as 0.05 for problems with more than two objectives. The initialization step of c-bensolve, where the initial outer approximation P^0 of the upper image $\mathcal{P}$ is formed, is done in three different ways for each algorithm, as described in Chapter 4. Then, in each iteration of c-bensolve, the current outer approximation is updated by intersecting it with halfspace H_k. In this step, we use three different procedures for the three algorithms and report the required CPU times in order to solve the same on-line vertex enumeration problem with each procedure. Note that for Algorithms 1 and 2, we call *onlinevert* for possibly different representations of the current outer approximation depending on the different initial polyhedrons, whereas for Algorithm 3 we call *onlinevert*2. In order to check the accuracy of Algorithms 1-3 we also employ *vert* from [46] for the same problem and confirm that the vertices of the updated polyhedron are found correctly.

Clearly, even the optimization problems of the same size may require different number of iterations of c-bensolve in order to be solved for a given error level ϵ. However, the number of iterations of c-bensolve is nothing but the number of halfspaces that define the polyhedral set to be generated. In other words, the 'size' of the induced vertex enumeration problem may be different for each multiobjective optimization problem of the same size. For our numerical tests, the aim is to compare the CPU times required to solve the vertex enumeration problem rather than the CPU times to solve the MOP.

In order to compare the average times that are spent for solving the vertex enumeration problem in each iteration, we consider a sub-sample of problems which require at least a certain number of iterations of c-bensolve. For example, for $n = 2$, $N = 5$ and $m = 10$, we generate hundred problems and solve them using c-bensolve where we employ three different vertex enumeration procedures within the iterations. Among hundred MOPs, the minimum number of iterations required is 3. Then, if we want to have a sample of hundred vertex enumeration problems of the same size, with the dimension being 2 and the number of defining halfspaces being 3, we need to consider only the first 3 iterations of these problems.

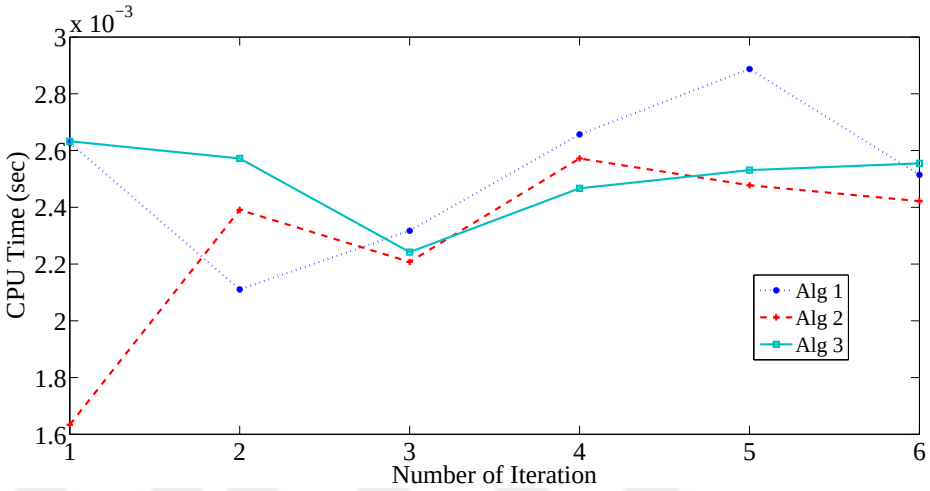

Figure 5.1: Run time performances of Algorithms 1-3 for problems with $n = 2$, $N = 5$, $m = 10$. The total number of iterations is 6 and the sample size is 45.

However, instead of considering hundred rather small-sized problems, we reduce the sample size such that it is at least forty and we increase the size of the vertex enumeration problem accordingly. The procedure is as follows. We list hundred problems according to the number of iterations that they require, in a non-increasing order. We check the number of iterations that the 40^{th} problem requires, say M. Then, we consider the sub-sample of problems which requires at least M iterations to be solved. Clearly, the sample size is at least forty. This way, we obtain a sample of vertex enumeration problems with dimension 2 and number of halfspaces to be intersected M. For $n = 2$, $N = 5$ and $m = 10$, we obtain $M = 6$ and the sample size is 45. Figure 5.1 show the average CPU time that is spent at each iteration of the vertex enumeration problem by each algorithm.

According to Figure 5.1, there is no clear distinction between the run times of the three algorithms. However, we observe that the first iteration of Algorithm 2 requires less time than the others.

For the rest of the numerical tests, we follow a similar procedure in order to determine the sample size and the size of the vertex enumeration problem. For two dimensional problems, we let the sample size to be at least 40, for three, four and five dimensional problems, we let the sample size to be at least 20.

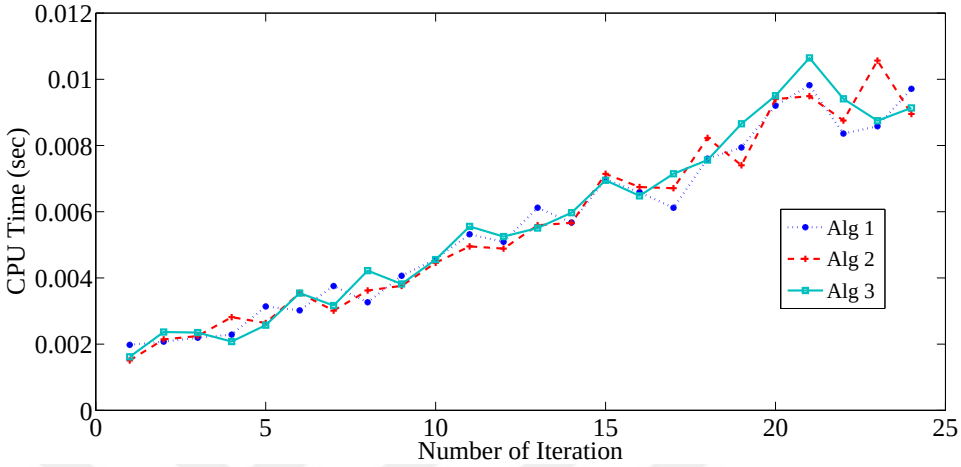

Figure 5.2: Run time performances of Algorithms 1-3 for problems with $n = 2$, $N = 30$, $m = 60$. The total number of iterations is 24 and the sample size is 41.

For $n = 2$, that is, for biobjective problems, we consider six set of parameters, where we take the number of variables N as 5, 10, 20, 30, 40, 50 and the number of constraints m as $2N$. For each set, we generate hundred problems and consider the sub-samples as explained above. The graph for $N = 30$ can be seen in Figure 5.2 and the graphs for the rest can be seen in Appendix, see Figures A.1-A.4.

Note that for 2-dimensional vertex enumeration problems, there is no clear distinction regarding the performances (CPU times) of the three algorithms. We observe that Algorithms 2 and 3 are better at the first few iterations. Note that this is expected as the initial polyhedron for Algorithm 1 has more 'artificial' vertices than the initial polyhedrons generated for the other two algorithms.

For $n = 3$, we consider five set of parameters, where we take the number of variables N as 5, 10, 20, 30, 40 and the number of constraints m as $2N$. For each set, we generate forty problems and consider the sub-samples of sizes at least 20. The graphs for $N = 10$ and $N = 30$ can be seen in Figures 5.3 and 5.4; and the graphs for the rest can be seen in Appendix, see Figures A.5-A.7.

As it can be seen from the figures, Algorithm 3 works better for three dimensional problems. Indeed, the average CPU times for each iteration that is spent by Algorithms 1 and 2 are around twice as the average run times that is spent by

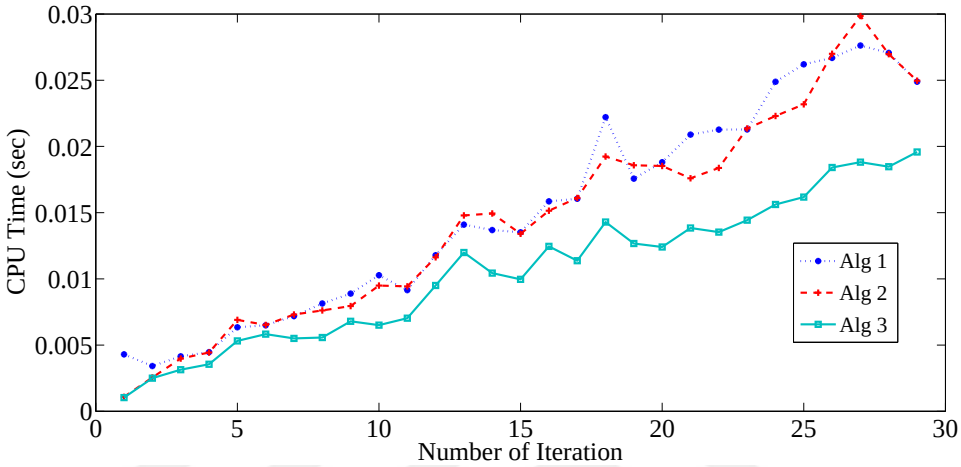

Figure 5.3: Run time performances of Algorithms 1-3 for problems with $n = 3$, $N = 10$, $m = 20$. The total number of iterations is 29 and the sample size is 21.

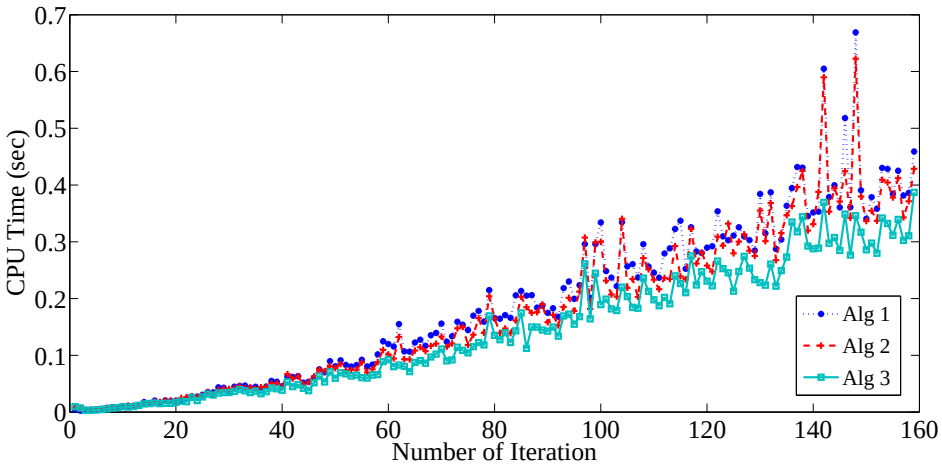

Figure 5.4: Run time performances of Algorithms 1-3 for problems with $n = 3$, $N = 30$, $m = 60$. The total number of iterations is 159 and the sample size is 21.

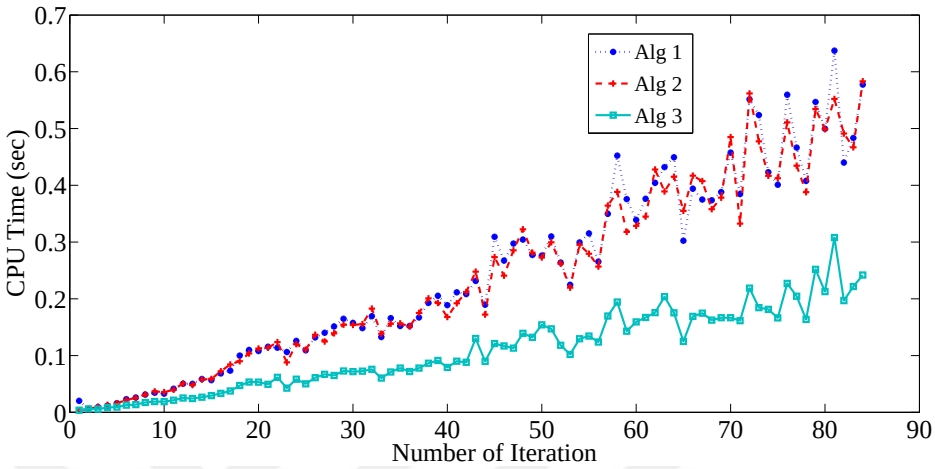

Figure 5.5: Run time performances of Algorithms 1-3 for problems with $n = 4$, $N = 10$, $m = 20$. The total number of iterations is 84 and the sample size is 20.

Algorithm 3. As before, the first couple of iterations of Algorithm 2 is faster than that of Algorithm 1; but for the rest of the iterations there is no clear distinction between the two.

For $n = 4$, we consider two sets of parameters, where we take the number of variables N as 5, 10 and the number of constraints m as $2N$. Again, we generate forty problems for each set and consider the sub-samples of sizes that are at least 20. The graphs for $N = 10$ can be seen in Figure 5.5. The graph for $N = 5$ can be seen in Appendix, Figure A.8.

Clearly, Algorithm 3 works better also for four dimensional vertex enumeration problems, whereas the performances of the other algorithms seem to be similar. Note that according to Figure 5.5, the average CPU times for each iteration that is spent by Algorithms 1 and 2 are even more than twice as the average run times that is spent by Algorithm 3, especially as the number of iterations increase. Note that this trend can also be observed by checking the graph for $N = 5$, $m = 10$, see Appendix, Figure A.8.

For $n = 5$, generate a set of problems where $N = 5$ and $m = 10$. The run time performances are shown in Figure 5.6.

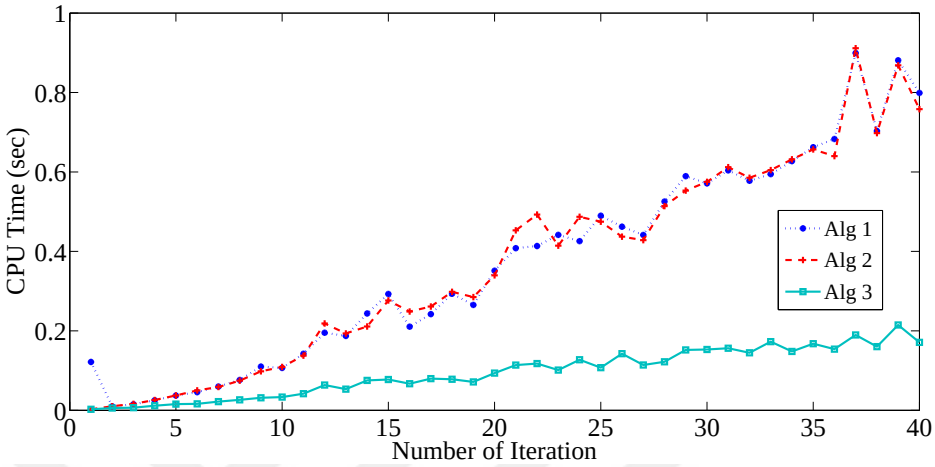

Figure 5.6: Run time performances of Algorithms 1-3 for problems with $n = 5$, $N = 5$, $m = 10$. The total number of iterations is 40 and the sample size is 20.

A similar trend is also observed for five dimensional problems. Algorithm 3 works much better than the other two algorithms. Indeed, by observing the graph, we see that CPU times for each iteration that is spent by Algorithms 1 and 2 are more than three times as the average run times that is spent by Algorithm 3. This can be observed clearly as the number of iterations increase.

By considering all the graphs that are generated, we also have some general observations regarding iterative vertex enumeration algorithms. First of all, we see that the CPU time spent for each iteration increases as the iteration number increases. This is expected since in general, the number of vertices to be checked in each iteration increases. Note that this is a trend which holds in general. Clearly, if a halfspace excludes many vertices of the previous polyhedron, the number of vertices may decrease in the next iteration and this explains why these graphs are not strictly increasing.

Secondly, we see that the CPU time increases as the dimension of the problems increases. This can be seen for instance by observing run time performances of problems with $N = 5$, $m = 10$ for different dimensions, see Figures 5.1, A.5, A.8, 5.6. Consider for instance the average times spent for the fifth iteration by Algorithm 3: for $n = 2$, it is around 2.5×10^{-3}; for $n = 3$, it is around 3×10^{-3}; for $n = 4$, it is around 0.01; and for $n = 5$ it is around 0.02. Clearly, the

same trend holds true for Algorithms 1 and 2. For larger problems, that is for the problems with higher number of iterations, this trend gets even more obvious. For instance the average times spent for the 30^{th} iteration by Algorithm 2 is around 0.04 for $n = 3$, around 0.08 for $n = 4$, and around 0.75 for $n = 5$. Clearly, the vertex enumeration problem gets more difficult to solve as the dimension of the problem increases, indeed it seems to be true that the difficulty gets accelerated by the dimension.

In addition to the average CPU times, we also keep the minimum and maximum CPU times that each algorithm spends in each iteration. Below, one can see Table 5.1 which shows this data for $n = 5$, $N = 5$ and $m = 10$. Note that the first column shows the iteration number; columns 2-4 show the average CPU time spent by Algorithms 1-3; columns 5-6 (7-8 / 9-10) show the minimum and the maximum CPU time of Algorithm 1 (2 / 3).

Similarly, the data for $n = 3, N = 40, m = 80$ can be seen in Appendix, Table A.1.

The tests are conducted on the same computer with system features Intel(R) Core(TM) i5 CPU- M 480-2.67 Ghz, 6.00 GB RAM, X64 Windows 10 and we utilize MATLAB R2013a.

Table 5.1: Run time performances for Algorithms 1-3 for problems with n=5, $N = 5$, $m = 10$. The total number of iterations is 40 and the sample size is 20.

I	Av-A1	Av-A2	Av-A3	Mn-A1	Mx-A1	Mn-A2	Mx-A2	Mn-A3	Mx-A3
1	0.1218	0.0026	0.0025	0.0219	0.4179	0.0017	0.0081	0.0015	0.0057
2	0.0101	0.0103	0.0057	0.0059	0.0287	0.0057	0.0241	0.0035	0.0103
3	0.0151	0.0163	0.0067	0.0069	0.0283	0.0072	0.0450	0.0039	0.0134
4	0.0252	0.0258	0.0116	0.0105	0.0542	0.0105	0.0497	0.0048	0.0349
5	0.0372	0.0374	0.0156	0.0111	0.0772	0.0109	0.1081	0.0043	0.0387
6	0.0456	0.0502	0.0162	0.0103	0.1145	0.0098	0.1446	0.0048	0.0428
7	0.0599	0.0586	0.0219	0.0182	0.1747	0.0179	0.1534	0.0069	0.0530
8	0.0763	0.0745	0.0263	0.0247	0.1687	0.0255	0.1731	0.0085	0.0551
9	0.1101	0.0981	0.0316	0.0230	0.6590	0.0219	0.3916	0.0078	0.1590
10	0.1067	0.1091	0.0333	0.0269	0.3387	0.0274	0.4518	0.0091	0.1025
12	0.1953	0.2185	0.0636	0.0234	1.0153	0.0234	1.2790	0.0078	0.3847
15	0.2928	0.2765	0.0775	0.0210	2.4094	0.0199	2.0212	0.0094	0.5583
20	0.3514	0.3397	0.0936	0.0846	0.7139	0.0859	0.7533	0.0206	0.2518
25	0.4898	0.4749	0.1074	0.0305	2.5995	0.0317	2.4744	0.0108	0.4802
30	0.5712	0.5750	0.1531	0.1223	2.1153	0.1107	2.1876	0.0237	0.6199
40	0.7989	0.7581	0.1711	0.0743	3.0036	0.0713	2.7754	0.0177	0.6616

Chapter 6

Conclusion

In this thesis, the vertex enumeration problem which has an important usage especially in continuous multiobjective optimization algorithms is considered. There are different approaches to solve the vertex enumeration problems and we focus on iterative algorithms which solve an on-line vertex enumeration problem in each iteration. The reason behind is that an on-line vertex enumeration is solved in each iteration of Benson-type linear and convex MOP algorithms.

We use the well known 'double description method' in order to solve the on-line vertex enumeration problem. Note that this method works where the given polyhedron is bounded. For Benson-type MOP algorithms however, we deal with unbounded polyhedrons. Hence, the first approach is to use sufficiently large bounded initial polyhedrons which is assumed to include the set of all vertices of the polyhedron whose H-representation is provided. Accordingly, we propose Algorithms 1 and 2.

Algorithm 1 is a general vertex enumeration algorithm which works for bounded and unbounded polyhedrons and returns the V-representation of a given polyhedron, that is, the vertices and the extreme directions that generate it.

We modify Algorithm 1 and propose Algorithm 2 for unbounded polyhedrons whose recession cone is known. Note that for Benson-type MOP algorithms, the

polyhedron that is to be generated is (the approximation of) the upper image of the problem and the upper image of a convex MOP is known to be unbounded. Indeed, its recession cone is the positive orthant. Algorithm 2 uses that information and starts with a smaller initial polyhedron. Then, it uses the double description method to iterate.

Next, we propose a variant (Algorithm 3) of Algorithm 2 where we again assume that the extreme directions of the given polyhedron are known. Different from the previous ones, the initial polyhedron for Algorithm 3 is not assumed to be bounded. Instead, it is an unbounded polyhedron with given extreme directions which are used directly in the computations.

The three algorithms are implemented using MATLAB software and performance tests are conducted for randomly generated linear MOPs. Note that the dimension of the vertex enumeration problem is the number of the objective function in the respective MOP. It has been shown that as the dimension increases, the vertex enumeration problem gets more difficult and requires more CPU time to be solved.

We also show that Algorithm 3 works more efficient than Algorithms 1 and 2 for problems in higher dimensions. For two dimensional ones, there is no clear distinction between the performances of the algorithms. For three dimensional problems, Algorithm 3 works much better that the others. To be more specific, the average CPU time spent by the other algorithms are around twice as the average CPU time spent by Algorithm 3. As the dimension increases, the efficiency of Algorithm 3 compared to Algorithms 1 and 2 improves even more. For five dimensional problems the average CPU time spent by Algorithm 1 and 2 are more than three times of the average CPU time that is spent by Algorithm 3.

When we compare the test results for Algorithms 1 and 2, we see that the performances are almost the same except a few iteration at the very beginning of the algorithms. That is expected as the initial polyhedron for Algorithm 2 has less number of vertices and faces than the one that is generated for Algorithm 1.

We see that using the extreme direction information directly in solving the vertex enumeration problem iteratively increases the efficiency significantly. A future work is to focus on the implementation of Algorithm 3 and make it even more efficient by using convenient data structures. We believe that implementing it using C or C++ instead of MATLAB would also improve the run-time performances. As the vertex enumeration problem is the bottleneck of solving linear and convex MOPs using Benson-type algorithms, this will also improve the efficiency of those.

Bibliography

[1] H. Greenberg, "An algorithm for determining redundant inequalities and all solutions to convex polyhedra," *Numerische Mathematik*, vol. 24, no. 1, pp. 19–26, 1975.

[2] V. Kuznetsov, "Algorithms for finding the general solution of a system of linear inequalities," *USSR Computational Mathematics and Mathematical Physics*, vol. 6, no. 2, pp. 197–205, 1966.

[3] N. V. Chernikova, "Algorithm for finding a general formula for the non-negative solutions of a system of linear equations," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 4, pp. 151–158, 1964.

[4] N. V. Chernikova, "Algorithm for finding a general formula for the non-negative solutions of a system of linear inequalities," *USSR Computational Mathematics and Mathematical Physics*, vol. 5, no. 2, pp. 228–233, 1965.

[5] L. Khachiyan, E. Boros, K. Borys, V. Gurvich, and K. Elbassioni, "Generating all vertices of a polyhedron is hard," in *Twentieth Anniversary Volume:*, pp. 1–17, Springer, 2009.

[6] D. Avis, D. Bremner, and R. Seidel, "How good are convex hull algorithms?, volume 7 of computational geometry: Theory and applications," 1997.

[7] M. E. Dyer and L. G. Proll, "An improved vertex enumeration algorithm," *European Journal of Operational Research*, vol. 9, no. 4, pp. 359–368, 1982.

[8] B. K. Schmidt and T. H. Mattheiss, "Computational results on an algorithm for finding all vertices of a poltyope," *Mathematical Programming*, vol. 18, no. 1, pp. 308–329, 1980.

[9] D. Avis, "Computational experience with the reverse search vertex enumeration algorithm," *Optimization Methods and Software*, vol. 10, no. 2, pp. 107–124, 1998.

[10] K. Fukuda and A. Prodon, "Double description method revisited," *Combinatorics and Computer Science*, pp. 91–111, 1996.

[11] M. Balinski, "An algorithm for finding all vertices of a convex polyhedral set," *Journal of the Society for Industrial and Applied Mathematics*, vol. 9, no. 1, pp. 72–88, 1961.

[12] W. Altherr, "An algorithm for enumerating all vertices of a convex polyhedron," *Computing*, vol. 15, no. 3, pp. 181–193, 1975.

[13] H. P. Benson, "An outer approximation algorithm for generating all efficient extreme points in the outcome set of a multiple objective linear programming problem," *Journal of Global Optimization*, vol. 13, no. 1, pp. 1–24, 1998.

[14] A. Hamel, A. Löhne, and B. Rudloff, "Benson type algorithms for linear vector optimization and applications," *Journal of Global Optimization*, vol. 59, no. 4, pp. 811–836, 2014.

[15] A. Löhne, B. Rudloff, and F. Ulus, "Primal and dual approximation algorithms for convex vector optimization problems," *Journal of Global Optimization*, vol. 60, no. 4, pp. 713–736, 2014.

[16] L. Csirmaz, "Using multiobjective optimization to map the entropy region," *Computational Optimization and Applications*, vol. 63, no. 1, pp. 45 – 67, 2016.

[17] P.-C. Chen, P. Hansen, and B. Jaumard, "On-line and off-line vertex enumeration by adjacency lists," *Operations Research Letters*, vol. 10, no. 7, pp. 403 – 409, 1991.

[18] A. Löhne and B. Weißing, "BENSOLVE: A free VLP solver, version 2.0.1," 2015.

[19] F. Ulus, "C-BENSOLVE: A free convex VOP solver," 2014.

[20] T. H. Matheiss and D. S. Rubin, "A survey and comparison of methods for finding all the vertices of convex polyhedral sets," *Mathematics of Operations Research*, vol. 5, no. 2, pp. 167–185, 1980.

[21] M. E. Dyer, "The complexity of vertex enumeration methods," *Mathematics of Operations Research*, vol. 8, no. 3, pp. 381–402, 1983.

[22] G. Dantzig, A. Orden, and P. Wolfe, "The generalized simplex method for minimizing a linear form under linear inequality restraints," *Pacific Journal of Mathematics*, vol. 5, no. 2, pp. 183–195, 1955.

[23] T. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall, "The double description method," in *Contributions to the Theory of Games II* (H. W. Kuhn and A. W. Tucker, eds.), Princeton University Press, 1953.

[24] M. Maňas and J. Nedoma, "Finding all vertices of a convex polyhedron," *Numerische Mathematik*, vol. 12, no. 3, pp. 226–229, 1968.

[25] T. Mattheiss, "An algorithm for determining irrelevant constraints and all vertices in systems of linear inequalities," *Operations Research*, vol. 21, no. 1, pp. 247–260, 1973.

[26] V. Klee, "Polytope pairs and their relationship to linear programming," *Acta Mathematica*, vol. 133, no. 1, pp. 1–25, 1974.

[27] M. E. Dyer and L. G. Proll, "An algorithm for determining all extreme points of a convex polytope," *Mathematical Programming*, vol. 12, no. 1, pp. 81–96, 1977.

[28] D. Avis and K. Fukuda, "A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra," *Discrete and Computational Geometry*, vol. 8, no. 3, pp. 295–313, 1992.

[29] J. Provan, "Efficient enumeration of the vertices of polyhedra associated with network LPs," *Mathematical Programming*, vol. 63, no. 1, pp. 47–64, 1994.

[30] T. Ottmann, S. Schuierer, and S. Soundaralakshmi, "Enumerating extreme points in higher dimensions," in *Annual Symposium on Theoretical Aspects of Computer Science*, pp. 562–570, Springer, 1995.

[31] K. Fukuda, T. M. Liebling, and F. Margot, "Analysis of backtrack algorithms for listing all vertices and all faces of a convex polyhedron," *Computational Geometry*, vol. 8, no. 1, pp. 1–12, 1997.

[32] D. Bremner, K. Fukuda, and A. Marzetta, "Primal-dual methods for vertex and facet enumeration," *Discrete and Computational Geometry*, vol. 20, no. 3, pp. 333–357, 1998.

[33] N. Chernikova, "Algorithm for discovering the set of all the solutions of a linear programming problem," *USSR Computational Mathematics and Mathematical Physics*, vol. 8, no. 6, pp. 282–293, 1968.

[34] H. Le Verge, *A note on Chernikova's algorithm*. PhD thesis, INRIA, 1992.

[35] H. Uzawa, "A theorem on convex polyhedral cones," in *Studies in Linear and Non-Linear Programming* (K. Hurwice and H. Uzawa, eds.), Stanford University Press, 1958.

[36] R. W. Stokes, "A geometric theory of solution of linear inequalities," *Transactions of the American Mathematical Society*, vol. 33, no. 3, pp. 782–805, 1931.

[37] B. F. Sherman, "A counterexample to Greenberg's algorithm for solving linear inequalities," *Numerische Mathematik*, vol. 27, no. 4, pp. 491–492, 1976.

[38] M. Dyer and L. Proll, "Eliminating extraneous edges in Greenberg's algorithm," *Mathematical Programming*, vol. 19, no. 1, pp. 106–110, 1980.

[39] D. Chand and S. Kapur, "An algorithm for convex polytopes," *Journal of the Association for Computing Machinery*, vol. 17, no. 1, pp. 78–86, 1970.

[40] K. Fukuda and V. Rosta, "Combinatorial face enumeration in convex polytopes," *Computational Geometry*, vol. 4, no. 4, pp. 191–198, 1994.

[41] B. Chazelle, "An optimal convex hull algorithm in any fixed dimension," *Discrete and Computational Geometry*, vol. 10, no. 1, pp. 377–409, 1993.

[42] J. Evans and R. Steuer, "A revised simplex method for linear multiple objective programs," *Mathematical Programming*, vol. 5, no. 1, pp. 54–72, 1973.

[43] P. Armand, "Finding all maximal efficient faces in multiobjective linear proramming," *Mathematical Programming*, vol. 61, no. 1, pp. 357–375, 1993.

[44] B. Rudloff, F. Ulus, and R. Vanderbei, "A parametric simplex algorithm for linear vector optimization problems," *Mathematical Programming*, vol. 163, no. 1-2, pp. 213–242, 2017.

[45] M. Ehrgott, L. Shao, and A. Schöbel, "An approximation algorithm for convex multi-objective programming problems," *Journal of Global Optimization*, vol. 50, no. 3, pp. 397–416, 2011.

[46] A. Löhne, "BENSOLVE: A free VLP solver, version 1.2.," 2012.

Appendix A

Additional Graphs and Tables for the Computational Performances of the Algorithms

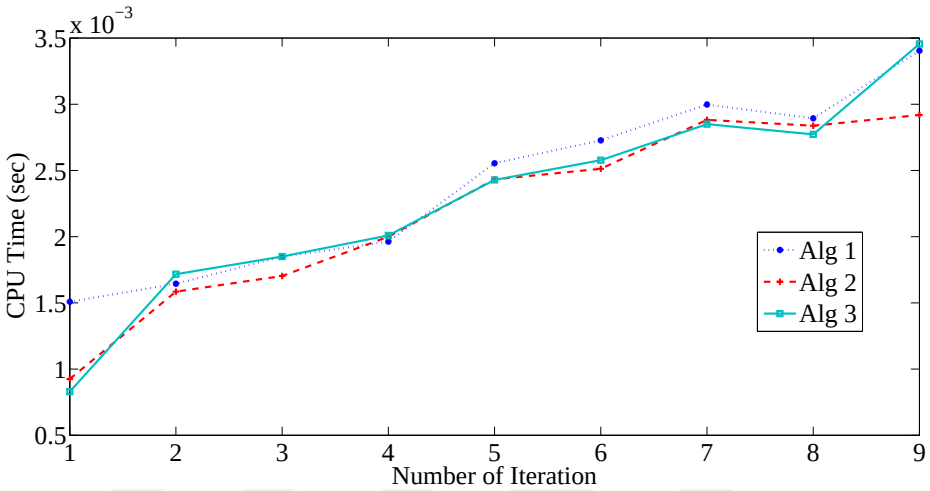

Figure A.1: Run time performances of Algorithms 1-3 for problems with $n = 2$, $N = 10$, $m = 20$. The total number of iteration is 9 and the sample size is 44.

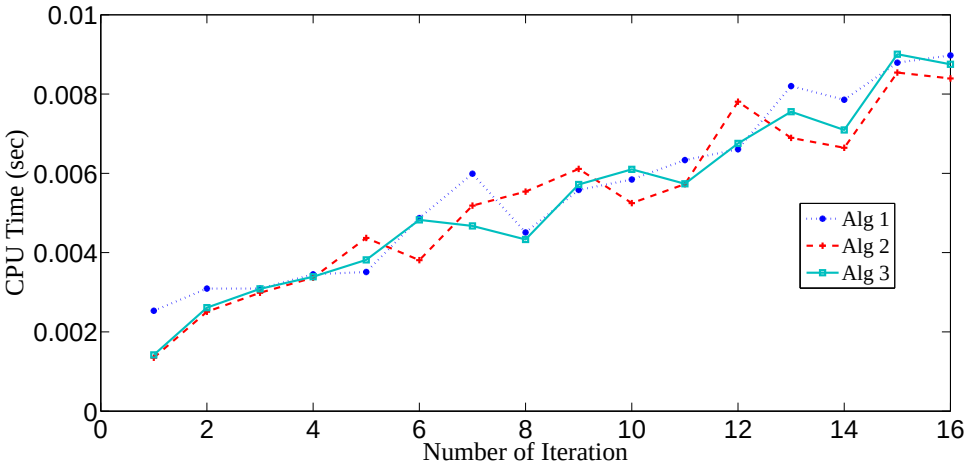

Figure A.2: Run time performances of Algorithms 1-3 for problems with $n = 2$, $N = 20$, $m = 40$. The total number of iterations is 16 and the sample size is 46.

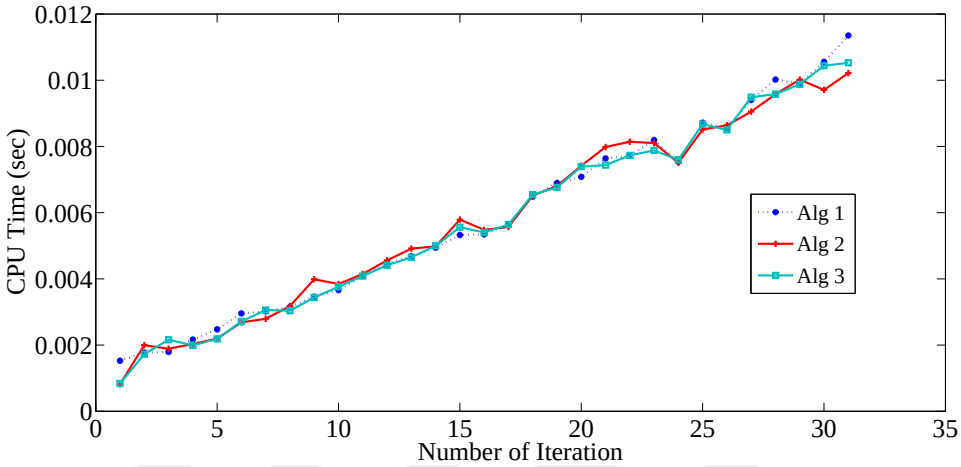

Figure A.3: Run time performances of Algorithms 1-3 for problems with $n = 2$, $N = 40$, $m = 80$. The total number of iteration is 31 and the sample size is 42.

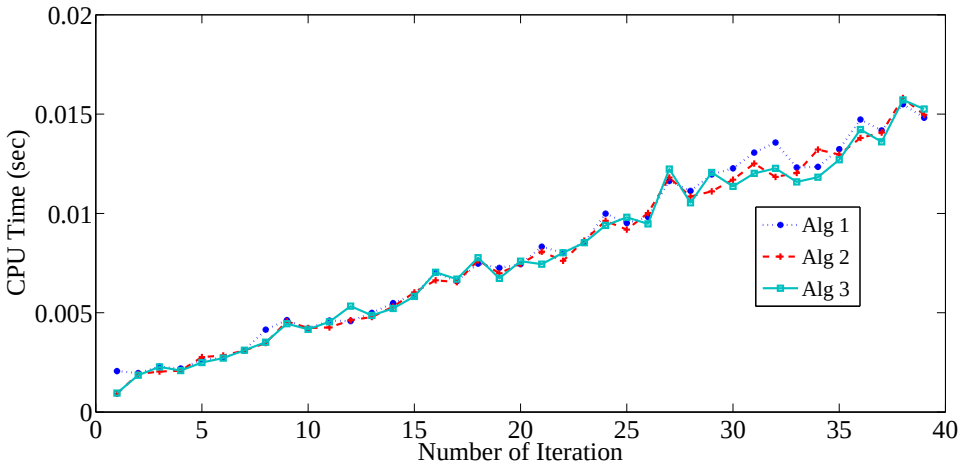

Figure A.4: Run time performances of Algorithms 1-3 for problems with $n = 2$, $N = 50$, $m = 100$. The total number of iterations is 39 and the sample size is 44.

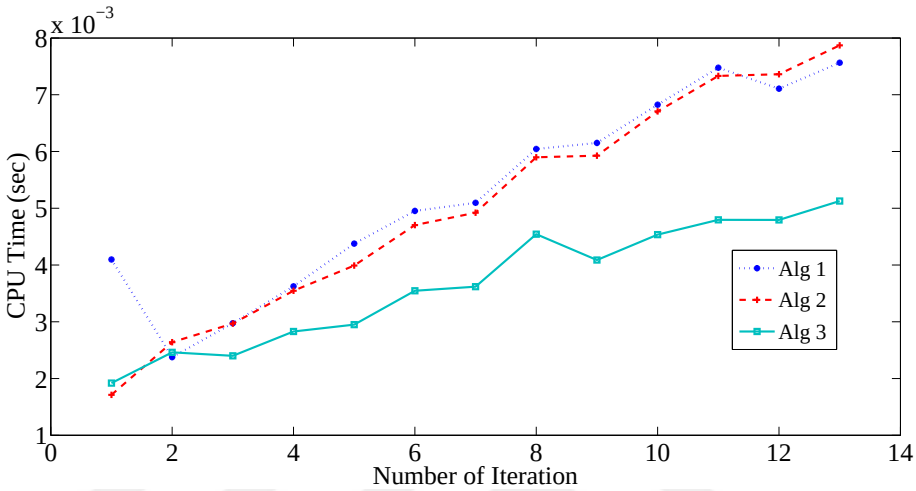

Figure A.5: Run time performances of Algorithms 1-3 for problems with $n = 3$, $N = 5$, $m = 10$. The total number of iterations is 13 and the sample size is 23.

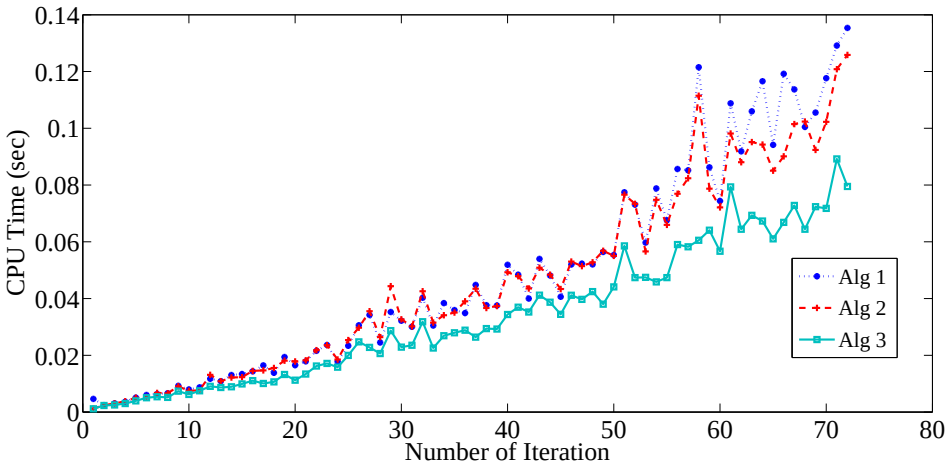

Figure A.6: Run time performances of Algorithms 1-3 for problems with $n = 3$, $N = 20$, $m = 40$. The total number of iterations is 72 and the sample size is 20.

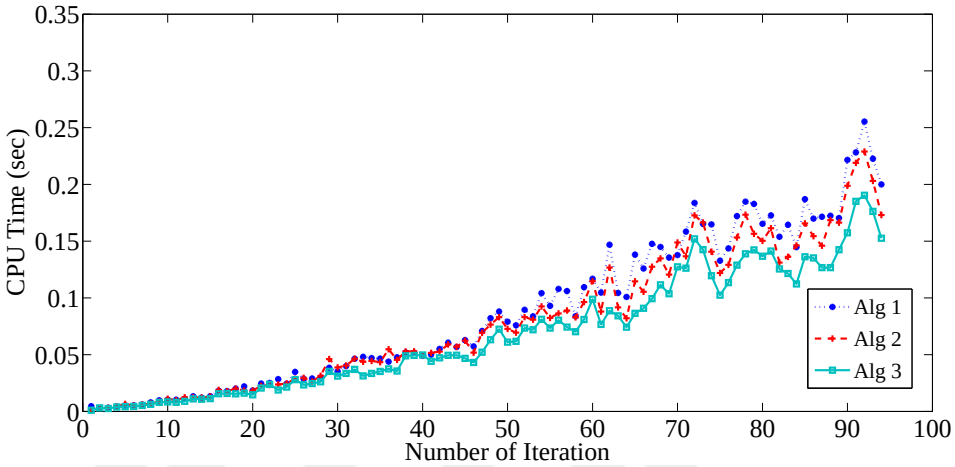

Figure A.7: Run time performances of Algorithms 1-3 for problems with $n = 3$, $N = 40$, $m = 80$. The total number of iterations is 94 and the sample size is 20.

Table A.1: Run time performances for Algorithms 1-3 for problems with n=3, $N = 40$, $m = 80$. The total number of iterations is 94 and the sample size is 20.

I	Mn-A1	Mx-A1	Mn-A2	Mx-A2	Mn-A3	Mx-A3
1	0.0021	0.0096	0.0008	0.0023	0.0008	0.0019
2	0.0017	0.0061	0.0016	0.0121	0.0017	0.0137
3	0.0025	0.0055	0.0024	0.0062	0.0019	0.0037
4	0.0025	0.0105	0.0024	0.0130	0.0022	0.0152
5	0.0027	0.0131	0.0027	0.0316	0.0025	0.0116
8	0.0041	0.0275	0.0042	0.0179	0.0036	0.0218
10	0.0055	0.0254	0.0056	0.0284	0.0049	0.0303
15	0.0094	0.0184	0.0091	0.0179	0.0061	0.0179
20	0.0116	0.0429	0.0115	0.0287	0.0094	0.0219
30	0.0191	0.0652	0.0189	0.0905	0.0158	0.0717
40	0.0291	0.0822	0.0262	0.0837	0.0221	0.2022
50	0.0379	0.1586	0.0461	0.1414	0.0297	0.1251
70	0.0743	0.2217	0.0757	0.2803	0.0577	0.2413
90	0.1207	0.4402	0.1058	0.5500	0.0992	0.3490

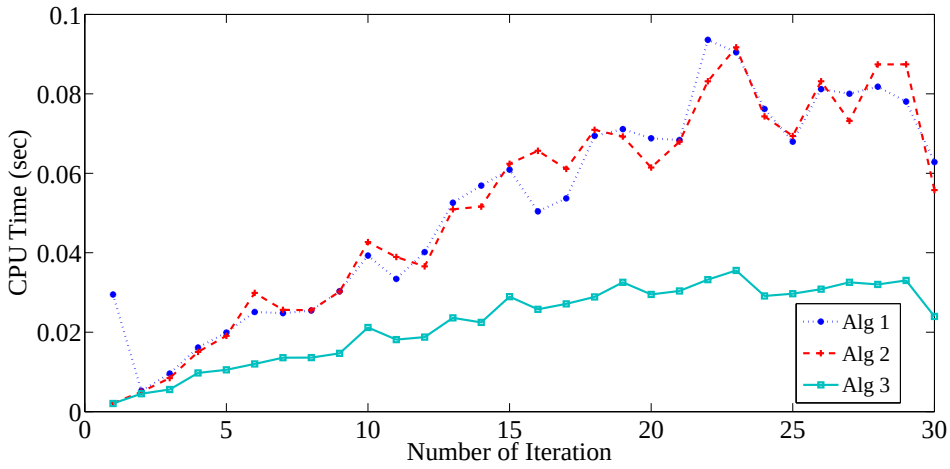

Figure A.8: Run time performances of Algorithms 1-3 for problems with $n = 4$, $N = 5$, $m = 10$. The total number of iterations is 30 and the sample size is 20.