OUTER APPROXIMATION ALGORITHMS FOR CONVEX VECTOR OPTIMIZATION PROBLEMS

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

İrem Nur Keskin

July 2021

OUTER APPROXIMATION ALGORITHMS FOR CONVEX VECTOR OPTIMIZATION PROBLEMS
By İrem Nur Keskin
July 2021

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)

Çağın Ararat

Gülşah Karakaya

Approved for the Graduate School of Engineering and Science:

[illegible]

Ezhan Karaşan V.
Director of the Graduate School

ABSTRACT

OUTER APPROXIMATION ALGORITHMS FOR CONVEX VECTOR OPTIMIZATION PROBLEMS

İrem Nur Keskin
M.S. in Industrial Engineering
Advisor: Firdevs Ulus
July 2021

There are different outer approximation algorithms in the literature that are designed to solve convex vector optimization problems in the sense that they approximate the upper image using polyhedral sets. At each iteration, these algorithms solve vertex enumeration and scalarization problems. The vertex enumeration problem is used to find the vertex representation of the current outer approximation. The scalarization problem is used in order to generate a weakly C-minimal element of the upper image as well as a supporting halfspace that supports the upper image at that point. In this study, we present a general framework of such algorithm in which the Pascoletti-Serafini scalarization is used. This scalarization finds the minimum 'distance' from a reference point, which is usually taken as a vertex of the current outer approximation, to the upper image through a given direction. The reference point and the direction vector are the parameters for this scalarization.

The motivation of this study is to come up with efficient methods to select the parameters of the Pascoletti-Serrafini scalarization and analyze the effects of these parameter selections on the performance of the algorithm. We first propose three rules to choose the direction parameter at each iteration. We conduct a preliminary computational study to observe the effects of these rules under various, rather simple rules for vertex selection. Depending on the results of the preliminary analysis, we fix a direction selection rule to continue with. Moreover, we observe that vertex selection also has a significant impact on the performance, as expected. Then, we propose additional vertex selection rules, which are slightly more complicated than the previous ones, and are designed with the motivation that they generate a well-distributed points on the boundary of the upper image. Different from the existing vertex selection rules from the literature, they do not require to solve additional single-objective optimization problems.

Using some test problems, we conduct a computational study where three different measures set as the stopping criteria: the approximation error, the runtime, and the cardinality of the solution set. We compare the proposed variants and some algorithms from the literature in terms of these measures that are used as the stopping criteria as well as an additional proximity measure, hypervolume gap. We observe that the proposed variants have satisfactory results especially in terms of runtime. When the approximation error is chosen as the stopping criteria, the proposed variants require less CPU time compared to the algorithms from the literature. Under fixed runtime, they return better proximity measures in general. Under fixed cardinality, the algorithms from the literature yield better proximity measures, but they require significantly more CPU time than the proposed variants.

Keywords: Multiobjective optimization, convex vector optimization, approximation algorithms.

ÖZET

DIŞ BÜKEY VEKTÖR OPTİMİZASYON PROBLEMLERİ İÇİN DIŞ YAKINSAMA ALGORİTMALARI

İrem Nur Keskin
Endüstri Mühendisliği, Yüksek Lisans
Tez Danışmanı: Firdevs Ulus
Temmuz 2021

Literatürde, polyhedral kümeler ile dışbükey vektör optimizasyonu problemlerinin üst görüntüsünü yakınsayan bazı algoritmalar bulunmaktadır. Bu algoritmalar, her yinelemede bir köşe noktası sıralama problemi, bir de skalarizasyon problemi çözerler. Köşe noktası sıralama problemi güncel dış yakınsak kümeyi oluşturan yarı-uzayların kesiştiği köşeleri bulmak için kullanılmaktadır. Skalarizasyon problemleri hem üst görüntü üzerinde bir C-minimal eleman bulmayı hem de üst görüntüye teğet bir yarı-uzay bulunmasını sağlamaktadır. Bu çalışmada, Pascoletti-Serafini skalarizasyonu kullanan bu tip algoritmalar için genel bir algoritma yapısı sunmaktayız. Bu skalarizasyon tekniği bir referans noktasından belli bir yönde ilerleyerek üst görüntü ile referans noktası arasındaki uzaklığı bulmaktadır. Referans noktası ve yön parametresi bu skalarizasyonun parametreleridir ve referans noktası genellikle dış yakınsak kümenin bir köşe noktasıdır.

Bu çalışmadaki ana motivasyonumuz Pascoletti-Serafini skalarizasyonunun parametrelerini verimli bir şekilde seçecek yöntemler geliştirmek ve bu seçimlerin algoritma verimliliği üzerindeki etkisini gözlemlemektir. Bu sebeple öncelikle yön parametresini seçmek için üç yöntem sunulmuş ve bunların algoritma üzerindeki etkisini inceleyen öncü bir hesaplama çalışması yapılmıştır. Bu öncü çalışmada köşe noktası seçimi için basit farklı yöntemler denenmiştir. Bu çalışma doğrultusunda ileride kullanılmak üzere bir yön parametresi sabitlenmiştir. Beklendiği gibi bu çalışma bize köşe noktası seçiminin de algoritmanın performansı üzerinde önemli bir etkiye sahip olduğunu göstermiştir. Bu sebeple üst görüntünün sınırında iyi dağılım gösterme potansiyeli olabilecek üç adet ek köşe noktası seçim yöntemi sunulmuştur. Bu yöntemler, literatürde bulunan köşe noktası seçimi yöntemlerinin aksine ek modeller çözmeyi gerektirmemektedir.

Farklı problem kümeleri üzerinde hata payı, toplam süre ve çözüm kümesinin eleman sayısı olmak üzere üç durma kriteri kullanılarak bir hesaplamalı çalışma yapılmıştır. Önerilen varyantlar literatürdeki bazı algoritmalarla bu durma kriterlerine göre karşılatırılmıştır. Ek olarak, başka bir yakınsama ölçüsü olan hiperhacim farkı da varyantların karşılaştırılmasında kullanılmıştır. Bu çalışmanın sonucunda önerilen varyantların tatmin edici sonuçlar verdiğini gözlemlemekteyiz. Hata payı durma kriteri olarak alındığında önerilen varyantlar, literatürdeki varyantlardan daha kısa sürede problemi çözmektedir. Sabit süre alındığındaysa yakınsama ölçüleri bakımından önerilen varyantlar genellikle daha iyi sonuçlar bulmaktadır. Sabit çözüm kümesi durma kriteri olarak alındığında literatürdeki algoritmalar genellikle problemleri daha iyi yakınsamakta ancak bu durumda çözüm süreleri önerilen varyantlardan daha uzun olmaktadır.

Anahtar sözcükler: Çok amaçlı optimizasyon, dışbükey vektör optimizasyonu, yakınsama algoritmaları.

Acknowledgement

I owe my deepest gratitude to my advisor Asst. Prof. Firdevs Ulus. This thesis would not be possible without her endless guidance and patience. I am indebted for the time she devoted to increase the quality of this research. She set me a great example of what a scholar should be not only with her wisdom and dedication but also with her mature personality. She is the best advisor that one can ask for.

I would like to extend my gratitude to Asst. Prof. Çağın Ararat and Asst. Prof. Gülşah Karakaya for accepting to read and review this thesis. I am indebted for their insightful comments. I would also like to thank the whole industrial engineering faculty. I learned a lot from each of them during my undergraduate and graduate years at Bilkent University.

I would like to thank my office mates İsmail Burak Taş, Efe Sertkaya, Duygu Söylemez, Deniz Akkaya, Ömer Ekmekçioğlu, and Aysu Özel. Although our interaction lasted short due to these extraordinary times, you contributed a lot to me. Many thanks to my friends from the "adjacent" office, Deniz Şimşek and Şifa Çelik, for making these two years much more fun and bearable. I would especially like to thank Emre Düzoylum for always making me smile, supporting me unconditionally, and for being my best friend throughout this experience.

I would like to thank Çağla Kaya, Hazal Hızoğlu, Bihter Bayrak, İlayda Kavak, Rümeysa Eren, Jülide Ateş and Ayça Çınar for being by my side since the beginning of high school. Even when we are apart, they always make me feel supported. I would also like to thank Hacer Korkut, Ece Önen, Simge Okur, Deniz Karazeybek, Büşra Dişli and Başak Dik for their invaluable friendship.

Last but not least, I would like to thank my parents Selda and Kemal, and my sister Ece for their endless support and for always believing in me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Multiobjective optimization problem (MOP) refers to optimizing more than one conflicting objectives simultaneously. For many problems in various fields from finance to engineering, several objectives maybe in trade-off which makes multiobjective optimization a useful tool. One such application from finance is portfolio optimization where there may be multiple objectives such as maximizing expected return and minimizing risk while selecting the best portfolio [1]. For a MOP with conflicting objectives, there is no single solution optimizing all. Instead, feasible solutions that cannot be improved in one objective without deteriorating in least another objective, namely *efficient solutions*, are of interest. The image of the efficient solutions are referred as *nondominated points* in the objective space.

An optimization problem that requires to minimize or maximize a vector-valued objective function with respect to a partial ordering induced by an ordering cone C is referred to as vector optimization problem (VOP). MOPs can be seen as special instances of VOPs where the ordering cone is the nonnegative orthant. The terminology in VOPs is different from MOPs: the efficient solutions in MOP are referred as *minimizers* for VOPs, and the nondominated points are called C-minimal points. Vector optimization is also widely used in different fields, see for instance [2], [3] for applications in financial mathematics and [4] for an application in stochastic optimization.

Among different types of VOPs, this thesis focuses on convex vector optimization problems (CVOPs). One of the most common approaches to "solve" VOPs is to solve scalarization problems, which are single objective optimization problems that are designed to find minimizers of the corresponding VOP. In general, a scalarization model is parametric and has the potential to generate a 'representative' set of minimizers when solved for different set of parameters. Through this thesis, two scalarization methods will be used. The first one is one of the most frequently used scalarization techniques, namely, the weighted sum scalarization [5], which is performed by optimizing the weighted sum of the objectives over the original feasible region for a weight vector from the dual of the ordering cone. Despite the application is simple, this technique fails to find some minimizers for non-convex problems. In addition, we use the Pascoletti-Serrafini scalarization [6], which aims to find the closest C-minimal point from a given reference point through a given direction parameter. Unlike weighted sum scalarization, it has a potential to find all minimizers of a given VOP.

Different algorithms are proposed to generate a 'representative' subset of C-minimal points of CVOPs in the literature [7, 8, 9, 10, 11]. We will mention these briefly in Section 2 and explain some of them in more detail in Section 5.5. The aim of these algorithms are to find polyhedral inner and outer approximations of the so called upper image of the CVOP (a set in the image space such that its boundary includes the set of all C-minimal points) with a certain proximity.

In general, these algorithms solve a vertex enumeration problem and a scalarization problem in each iteration. More specifically, most of these algorithms solve a Pascoletti-Serafini scalarization in each iteration. As mentioned above, this scalarization utilizes two parameters: a reference point and a direction vector, both in the objective space.

In this thesis, we present a general framework of an outer approximation algorithm to solve CVOPs from the literature. We consider different rules to select the two parameters of the scalarization problem. We first propose different direction selection rules, and conduct a preliminary computational study to compare them. Since the selection of the reference point may also affect the performance of the algorithm, for this preliminary analysis, we also consider different 'simple' rules for that.

In addition to the selection of direction parameter, we consider different rules for selecting the reference point for the scalarization. As briefly explained above, the algorithm that is described in this thesis finds polyhedral outer approximations to the upper image. Then, the vertices of the current outer approximation are considered as candidate reference points for the scalarization problem to be solved in the next iteration. There are different approaches for selecting a vertex in the literature [7, 10] and this also has an impact on the performance of the algorithm.

We propose three vertex selection rules. The first one benefits from the clustering of the vertices. In particular, every vertex is assigned to one of the clusters that are formed at the beginning of the algorithm and in each iteration, the algorithm alters the cluster that the vertex is chosen from. The second rule selects the vertices by using the adjacency information among the vertices. The last vertex selection rule works only for convex multiobjective optimization problems. For this one, we employ a procedure which creates local upper bounds to the nondominated points. Then, the vertex selection is done by using the distances between the vertices and these upper bounds.

Together with the variants that are proposed in this thesis, we also implement three relevant algorithms from the literature and we provide an extensive computational study to observe the behavior of them under different stopping conditions.

This thesis is organized as follows. In Chapter 2, we present a review of the relevant algorithms from the literature. In Chapter 3, we provide the preliminaries and the notation used throughout the study. In Chapter 4, we provide the solution concepts, and the scalarization methods used inside the algorithm. In Chapter 5, we present an outer approximation algorithm to solve CVOP and we construct different variants of this algorithm by fixing rules for direction and vertex selections. In Chapter 6, we present a computational study under three stopping criteria: approximation error, CPU time and cardinality. We compare the proposed variants with the algorithms from the literature. Finally, we conclude the thesis in Chapter 7.

Chapter 2

Literature Review on CVOP Algorithms

There are iterative algorithms in literature that utilize the scalarization methods to solve convex VOPs in the sense that they find an approximation to the set of all C-minimal points in the objective space. To the best of our knowledge, the first such algorithm is designed to solve linear multiobjective optimization problems and proposed in 1998, by Benson [12]. This algorithm generates the set of all nondominated points of linear MOPs by obtaining improved polyhedral outer approximations of it iteratively. Later, this algorithm is generalized to solve linear VOPs; moreover, using geometric duality results, a geometric dual counterpart of this algorithm is also proposed, see [13].

In 2011, Benson's algorithm for linear problems is extended by Ehrgott et al. [8] so that it can solve convex VOPs. Then, in 2014, Löhne et al. [9] proposed another variant of this algorithm which solves less number of optimization problems in each iteration and proposed also a geometric dual variant to it.

On the other hand, in 2003, Klamroth et al. [7] proposed outer and inner approximation algorithms for convex and non-convex MOPs. The mechanism of the outer approximation algorithm for the convex case is similar to Benson's algorithm for the linear MOPs from [12]. The main difference is that they use a Gauge-based scalarization method to choose a reference point and find a non-dominated solution corresponding to that reference point. This method measures a distance in the objective space using polyhedral gauge functions. We will show in Section 5.5 that this model can be equivalently expressed as a Pascoletti-Serafini scalarization with specific parameters. Hence, the outer approximation algorithm for convex MOPs from [7] can be seen as an extension of the linear MOP algorithm from [12]. Different from [12], in this algorithm, the selection for the reference point for the scalarization is not arbitrary but depends on a specific rule which will be detailed in Section 5.5. Note that in [7], there is also an inner approximation algorithm for the convex problems and it solves weighted sum scalarization problems in each iteration. Indeed, the design of the geometric dual algorithm from [9] and the inner approximation algorithm from [7] are similar. However, they are described from different points of views. A final remark for the outer and inner approximation algorithms for convex MOPs from [7] is that the convergence rate of the algorithms are provided for the biobjective case.

Recently in 2021, Dörfler et al. [10] proposed a variant for Benson's algorithm for convex VOPs that includes a vertex selection procedure which also yields a direction parameter for the Pascoletti-Serafini scalarization. It benefits from the current inner approximation to compute these parameters. More specifically, in each iteration it picks the vertex of the current outer approximation that yields the maximum distance to the current inner approximation. In order to pick that vertex, the algorithm solves quadratic programming problems for the vertices of the current outer approximation.

In addition to the algorithms which solve Pascoletti-Serafini scalarization (or equivalent models) in each iteration, recently Ararat et. al [11] propose an outer approximation algorithm for CVOPs which solves norm-minimizing scalarizations instead. This scalarization does not require a direction parameter but only requires a reference point, which is again selected among the vertices of the current outer approximation. Different from the other outer approximation algorithms for CVOPs, in [11] the finiteness of the algorithm is shown.

The properties of the algorithms mentioned above are summarized in Table 2.1. Note that these algorithms differ in terms of the selection procedures for the parameters of the Pascoletti-Serafini scalarization. The algorithms in [7, 10], require solving additional models in order to select a vertex at each iteration. This feature enables them to provide the current approximation error at each iteration of the algorithm at the cost of solving considerable amount of models. The rest of the algorithms do not solve additional models and they provide the approximation error after the termination. For the selection of the direction parameter, [7, 8] use a fixed point from the upper image; [10] uses the point from the inner approximation that yield the minimum distance to the selected vertex; and [9] uses a fixed direction from the interior of the ordering cone through the algorithm.

Table 2.1: Existing Outer Approximation Algorithms to Solve Convex VOP

Algorithm	**Finiteness / Convergence**	**Choice of Direction**	**Vertex Selection (VS)**	**Models Solved for VS**	**Approximation Error**
Klamroth et al. [7]	Convergence for biobjective	Inner point (fixed)	Distance to upper image	Gauge-based Model	At each iteration
Ehrgott et al. [8]	-	Inner point (fixed)	Arbitrary	-	After Termination
Löhne et al. [9]	-	Fixed	Arbitrary	-	After Termination
Dörfler et al. [10]	-	Inner point (changing)	Distance to inner approximation	Quadratic Model	At each iteration
Ararat et al. [11]	Finiteness	Not Relevant	Arbitrary	-	After Termination

Chapter 3

Preliminaries

Let S be a subset of $\mathbb{R}^p$. The convex hull, conic hull, interior, closure and boundary of S is denoted by $\operatorname{conv} S$, $\operatorname{cone} S$, $\operatorname{int} S$, $\operatorname{cl} S$ and $\operatorname{bd} S$ respectively. $z \in \mathbb{R}^p \setminus \{0\}$ is a *(recession) direction* of S, if $y + \gamma z \in S$ for all $\gamma \geq 0$ and $y \in S$. The set of all recession directions of S is the recession cone of S and denoted by $\operatorname{recc} S$.

Let $S \subseteq \mathbb{R}^p$ be convex and $F \subseteq S$ be a convex subset. If $\lambda y^1 + (1-\lambda) y^2 \in F$ for some $0 < \lambda < 1$ holds only if both y^1 and y^2 are elements of F, then F is a *face* of S. A zero dimensional face is an *extreme point*, a one dimensional face is an *edge* and a $p-1$ dimensional face is a *facet* of S, see [14]. A recession direction $z \in \mathbb{R}^p \setminus \{0\}$ of convex set S is said to be an *extreme direction* of S, if $\{v + rz \in \mathbb{R}^p |\ r \geq 0\}$ is a face for some extreme point v of S.

Let $S, T \subseteq \mathbb{R}^p$. The Hausdorff distance between S and T is

$$d_H(S,T) = \max\{\sup_{s \in S} d(s,T), \sup_{t \in T} d(t,S)\}$$

where $d(t,S) := \inf_{s \in S} ||t - s||$ denotes the distance of point $t \in \mathbb{R}^p$ to set S, and $\|\cdot\|$ is a norm on $\mathbb{R}^p$. If not specified, we assume that $\|\cdot\|$ is the Euclidean norm.

Proposition 3.0.1 ([11]). *If $S \supseteq T$ and S is a polyhedral set such that $\operatorname{recc} S = \operatorname{recc} T$, then $d_H(S,T) = \max_{v \in V^S} d(v,T)$ where V^S is the set of all vertices of S.*

The Minkowski sum of sets S and T are $S + T = \{s + t \in \mathbb{R}^p | \ s \in S, t \in T\}$. By $S - T$, we denote the set $S + (-1) \cdot T = \{s - t \in \mathbb{R}^p | \ s \in S, t \in T\}$. Moreover, we have for $\alpha \in \mathbb{R}$, $\alpha S = \{\alpha s \in \mathbb{R}^p | \ s \in S\}$.

Let $C \subseteq \mathbb{R}^p$ be a convex pointed cone, that is, it contains no lines. C defines a partial ordering $\leq_C$ by $y^1 \leq_C y^2$ holds if and only if $y^2 - y^1 \in C$. $C^+ := \{a \in \mathbb{R}^p | \ a^\top x \geq 0, \forall x \in C\}$ is the dual cone of C. C^+ is a closed convex cone.

Let $f : \mathbb{R}^n \to \mathbb{R}^p$ be a function. For a given convex pointed cone $C \subseteq \mathbb{R}^p$, f is said to be C-convex if $f(\lambda x + (1 - \lambda)y) \leq_C \lambda f(x) + (1 - \lambda)f(y)$ for all $x, y \in \mathbb{R}^p$.

If S is a *polyhedral convex set*, it is of the form

$$S = \{y \in \mathbb{R}^p \mid A^T y \geq b\} = \bigcap_{i=1}^{k} \{y \in \mathbb{R}^p \mid a_i^T y \geq b_i\}, \tag{3.1}$$

where $A \in \mathbb{R}^{p \times k}$, $b \in \mathbb{R}^k$, $a_i \in \mathbb{R}^p$ is the i^{th} column of matrix A and $b_i \in \mathbb{R}$ is the i^{th} component of b. The representation given in (3.1) is called the *halfspace representation* of S. On the other hand, if S has at least one extreme point (vertex), it can also be represented as the convex hull of all its vertices added to the conic hull of all its extreme directions, that is,

$$S = \operatorname{conv} V + \operatorname{cone} \operatorname{conv} D, \tag{3.2}$$

where V is the finite set of all vertices of S and D is the finite set of all extreme directions of S. The representation given by (3.2) of S is called the *vertex representation* of the polyhedral convex set S. The problem of finding the *vertex representation* of a set given its *halfspace representation* is called the *vertex enumeration problem.*

There are different approaches/solvers to solve a vertex enumeration problem. In this thesis, employ the MATLAB toolbox *bensolve tools* for this purpose [15, 16]. When one solves a vertex enumeration problem, *bensolve tools* yields the sets V, D as well as the set of all adjacent vertices $v_{adj} \subseteq V$ of each vertex $v \in V$. Note that two vertices are said to be *adjacent* if the line segment between them is an edge of S.

Throughout, we use the following notation: $e := (1, \ldots, 1)^\top \in \mathbb{R}^p$, and $e^j \in \mathbb{R}^p$ is the unit vector with j^{th} component being 1. $\mathbb{R}^p_+ := \{y \in \mathbb{R}^p | \ y \geq 0\}$, $\mathbb{R}^p_{++} := \{y \in \mathbb{R}^p | \ y_i > 0 \text{ for } i = 1, \ldots, p\}$.

Chapter 4

The Problem

We consider a convex vector optimization problem given by

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

where C is a convex pointed ordering cone with nonempty interior, $\mathcal{X} \subseteq \mathbb{R}^n$ is a nonempty closed convex set and $f : \mathbb{R}^n \to \mathbb{R}^p$ is a continuous C-convex function given by $f(x) := (f_1(x), \dots, f_p(x))^\top$. $f(\mathcal{X}) := \{f(x) \in \mathbb{R}^p |\ x \in \mathcal{X}\}$ is the image of $\mathcal{X}$ under f.

Assumption 4.0.1. *Throughout, we assume that C is polyhedral and $\mathcal{X}$ is compact with nonempty interior.*

4.1 Solution Concepts for (P)

Below we list the common terminology and the solution concepts for (P).

Definition 4.1.1. *Let $\mathcal{Y} \subseteq \mathbb{R}^p$ and $y \in \mathcal{Y}$. For a convex pointed cone $C \subseteq \mathbb{R}^p$, $y \in \mathcal{Y}$ is called C-minimal element of $\mathcal{Y}$ if $(\{y\} - C \setminus \{0\}) \cap \mathcal{Y} = \emptyset$. If C has nonempty interior, then $y \in \mathcal{Y}$ is called weakly C-minimal element of $\mathcal{Y}$ if $(\{y\} - \operatorname{int} C \setminus \{0\}) \cap \mathcal{Y} = \emptyset$. The set of all C-minimal (weakly C-minimal) elements of $\mathcal{Y}$ is denoted by $\operatorname{Min}_C \mathcal{Y}$ ($w\operatorname{Min}_C \mathcal{Y}$).*

Definition 4.1.2. *A feasible point* $\bar{x} \in \mathcal{X}$ *of* (P) *is called (weak) minimizer, if* $f(x)$ *is (weakly)* C*-minimal element of* $f(\mathcal{X})$*. The set of all minimizers (weak minimizers) is denoted by* $\mathcal{X}_E$ $(\mathcal{X}_{WE})$*.*

Definition 4.1.3. *For* $C = \mathbb{R}^p_+$*, the point* $y^I \in \mathbb{R}^p$ *is the* ideal point *of* (P) *if* $y^I_i := \inf\{f_i(x)|\ x \in \mathcal{X}\}$ *for* $i = 1, \dots, p$*. The point* $\hat{p} \in \mathbb{R}^p$ *is the* nadir point *of* (P) *if* $\hat{p}_i := \sup\{f_i(x)|\ x \in \mathcal{X}_E\}$ *for* $i = 1 \dots, p$*.*

We consider the set $\mathcal{P} := \operatorname{cl}(f(\mathcal{X}) + C)$, namely, the *upper image* for problem (P). For a convex VOP, the upper image is a closed convex set and the set of all weakly C-minimal points of $\mathcal{P}$ is $\operatorname{bd}\mathcal{P}$, see for instance [8]. Then, the set of weak minimizers of (P) is $\{x \in \mathbb{R}^p|\ f(x) \in \operatorname{bd}\mathcal{P} \cap f(\mathcal{X})\}$.

Remark 4.1.4. *Under Assumption 4.0.1,* $\mathcal{P} = f(\mathcal{X}) + C$*. This can be seen as follows: Let* z *be a limit point of* $\mathcal{P} = \operatorname{cl}(f(\mathcal{X}) + C)$*. Then, there exist* $y_n \in f(\mathcal{X})$ *and* $r_n \in C$ *for* $n \in \mathbb{N}$ *such that*

$$z = \lim_{n\to\infty} (y_n + r_n)$$

Since $\mathcal{X}$ *is compact and* f *is continuous, then* $f(\mathcal{X})$ *is compact. Hence,* (y_n) *is a bounded sequence and there exists a convergent subsequence* (y_{n_k}) *such that* $\lim_{k\to\infty} y_{n_k} =: \bar{y} \in f(\mathcal{X})$*. Say* $\bar{y} = f(\bar{x})$ *for* $\bar{x} \in \mathcal{X}$*. Note that* $\lim_{k\to\infty}(y_{n_k} + r_{n_k}) = z \in \mathbb{R}^p$*. Then,* $\lim_{k\to\infty}(y_{n_k} + r_{n_k}) - \lim_{k\to\infty} y_{n_k} = \lim_{k\to\infty} r_{n_k} =: \bar{r} \in \mathbb{R}^p$*. Since* C *is closed,* $\bar{r} \in C$*. Hence,* $f(\bar{\mathcal{X}}) + C = \operatorname{cl}(f(\bar{\mathcal{X}}) + C)$*.*

(P) is said to be *bounded* if $\mathcal{P} \subseteq \{y\} + C$ for some $y \in \mathbb{R}^p$. If $\mathcal{X}$ is compact, then (P) is bounded [9]. Note that in multiobjective setting where $C = \mathbb{R}^p_+$, the problem (P) is bounded if $y^I \in \mathbb{R}^p$. If the feasible region is compact, then there exists $x^i \in \mathcal{X}$ such that $y^I_i = f(x^i)$ for all $i = 1, \dots, p$. Hence, the ideal point can be computed by solving p single objective optimization problems. However, computing the nadir point is not always easy or even possible.

Below, we provide two solution concepts for bounded convex vector optimization problems. These are motivated from a set-optimization point of view. For details on such solution concepts, see for instance [13].

The first solution concept depends on a fixed direction parameter $c \in \operatorname{int} C$. It is introduced in [9] as 'finite ϵ-solution'. Here we add the term 'with respect to c' in order to emphasize this dependence.

Definition 4.1.5 ([9, Definition 3.3.]). *Let $c \in \operatorname{int} C$ be fixed, $\bar{\mathcal{X}} \subseteq \mathcal{X}$ be a nonempty finite set consisting of (weak) minimizers. If*

$$\operatorname{conv} f(\bar{\mathcal{X}}) + C - \epsilon\{c\} \supseteq \mathcal{P}$$

holds, then $\bar{\mathcal{X}}$ is called a finite (weak) ϵ-solution with respect to c.

In [11], an algorithm is proposed in order to approximate the upper image of a convex vector optimization problem and the following definition is used.

Definition 4.1.6 ([10],[11]). *A nonempty finite set $\bar{\mathcal{X}} \subseteq \mathcal{X}$ of (weak) minimizers is a* finite (weak) ϵ-solution *if*

$$\operatorname{conv} f(\bar{\mathcal{X}}) + C + B(0, \epsilon) \supseteq \mathcal{P}.$$

Note that both of these solution concepts yield an outer approximation to the upper image as well as an inner approximation ($\operatorname{conv} f(\bar{\mathcal{X}}) + C$) to it. The Hausdorff distance between these sets are bounded [10]. Moreover, if the feasible region $\mathcal{X}$ is compact, then for any $\epsilon > 0$, there exists a finite weak ϵ-solution (with respect to $c \in \operatorname{int} C$) to problem (P), see [9, Proposition 4.3] and [11, Proposition 3.8].

4.2 Scalarization Models and Related Results

There are different ways of generating (weak) minimizers to (P). One way is to solve scalarization models, that is, single objective optimization models based on some design parameters. Below, we provide two well-known ones together with some results regarding them.

The weighted sum scalarization model is given by

$$\begin{aligned} &\text{minimize } w^\top f(x) && \text{(WS}(w)) \\ &\text{subject to } x \in \mathcal{X}, \end{aligned}$$

where $w \in \mathbb{R}^p$ is the weight parameter.

Proposition 4.2.1 ([17, Corrolary 5.29])**.** *An optimal solution $x \in \mathcal{X}$ of* (WS(w)) *is a weak minimizer of* (P) *if* ($w \in C^+ \setminus \{0\}$)*. Conversely, for any weak minimizer $x \in \mathcal{X}$, there exists* ($w \in C^+ \setminus \{0\}$) *such that x is an optimal solution of* (WS(w))*.*

The next lemma is also related to (WS(w)) model and it will be used later.

Lemma 4.2.2. *If $\inf_{p\in\mathcal{P}} w^\top p \in \mathbb{R}$ for some $w \in \mathbb{R}^p$, then $w \in C^+$.*

Proof. Let $\beta < \inf_p w^\top p \in \mathbb{R}$ and assume to the contrary that $w \notin C^+$. Then, there exists $c \in C$ such that $w^\top c < 0$. As $c \in C$, we have $\lambda c \in C$ for any $\lambda \in \mathbb{R}_+$. Take $y \in f(\mathcal{X})$. As $y + \lambda c \in \mathcal{P}$, we have $w^\top(y + \lambda c) = w^\top y + \lambda w^\top c > \beta$ for any $\lambda \in \mathbb{R}_+$. This is a contradiction, since $w^\top c < 0$ and we can find a sufficiently large λ such that $w^\top y + \lambda w^\top c < \beta$. Hence $w \in C^+$. □

Pascoletti-Serafini scalarization model [6] is given by

$$\begin{aligned} &\text{minimize } z && \text{(PS}(v,d)) \\ &\text{subject to } f(x) \leq_C v + zd \\ &\qquad x \in \mathcal{X} \\ &\qquad z \in \mathbb{R}, \end{aligned}$$

where the parameters $v, d \in \mathbb{R}^p$ are referred to as the reference point and the direction, respectively. Below, we provide some well-known results and prove some further ones regarding the scalarization model.

Proposition 4.2.3. *[9, Proposition 4.5] If $(x^*, z^*) \in \mathbb{R}^{n+1}$ is an optimal solution of problem* (PS(v,d))*, then x^* is a weak minimizer . Moreover, $v + z^*d \in \operatorname{bd} \mathcal{P}$.*

Remark 4.2.4. *Suppose Assumption 4.0.1 holds. Then for all $z \in \mathbb{R}$, $f(x) \leq_C v + zd$ for some $x \in \mathcal{X}$ if and only if $v + zd \in \mathcal{P}$. To see, assume $f(x) \leq_C v + zd$ holds for some $x \in \mathcal{X}$, that is, $f(x) - v - zd \in C$ holds. Then, we have $v + zd \in \{f(x)\} + C \subseteq f(\mathcal{X}) + C \subseteq \mathcal{P}$. The other implication follows by Remark 4.1.4. Let $z \in f(\mathcal{X}) + C = \mathcal{P}$, then there exists $x \in \mathcal{X}$ such that $v + zd \in f(x) + C$ which implies $f(x) \leq_C v + zd$.*

Proposition 4.2.5. *Let $d \in \operatorname{int} C$ and $v \notin \mathcal{P}$. If (x^*, z^*) is an optimal solution for* (PS(v, d))*, then $z^* \geq 0$ and $z^* \|d\| \geq d(v, \mathcal{P})$.*

Proof. Let us assume to the contrary that $z^* < 0$. Note that $-z^*d \in \operatorname{int} C$, as $-z^* > 0$ and $d \in \operatorname{int} C$. Moreover, since (x^*, z^*) is feasible for (PS(v, d)), this implies $-f(x^*) + v \in \{-z^*d\} + C \subseteq C$. Hence, $v \in \{f(x^*)\} + C$. As it is given that $v \notin \mathcal{P}$, this is a contradiction. Hence, we conclude $z^* \geq 0$. Note that $y^* := v + z^*d \in \mathcal{P}$, [9]. Then, $d(v, \mathcal{P}) \leq \|y^* - v\| = z^* \|d\|$ holds. □

Proposition 4.2.6. *Suppose Assumption 4.0.1 holds. Let $v \in \mathbb{R}^p$. If $d \in \operatorname{int} C$, then there exists an optimal solution to* (PS(v, d))*.*

Proof. First, we show that there exists a feasible solution to PS(v, d) when $d \in \operatorname{int} C$. Suppose to the contrary that there does not exists such (x, z) pair which is feasible for the problem. Then, for any feasible $z \in \mathbb{R}$, we have $v + zd \notin \mathcal{P}$. Let us define

$$\mathcal{L} = \{v + zd, z \in \mathbb{R}\}$$

Clearly, $\mathcal{L}$ is a closed subset of $\mathbb{R}^p$. Moreover, $\mathcal{L} \cap \mathcal{P} = \emptyset$ holds. Then by hyperplane separation theorem, there exists $w \in R^p$ such that

$$w^\top (v + zd) \leq \inf_{p \in \mathcal{P}} w^\top p$$

for all $v + zd \in \mathbb{R}^p$. Indeed, by Lemma 4.2.2, $w \in C^+$. Note that the following also holds

$$\inf_{p \in \mathcal{P}} w^\top p \leq \inf_{x \in \mathcal{X}, c \in C} w^\top (f(x) + c) = \inf_{x \in \mathcal{X}} w^\top f(x) + \underbrace{\inf_{c \in C} w^\top c}_{0} = \inf_{x \in \mathcal{X}} w^\top f(x)$$

Hence, $w^\top v + zw^\top d \leq \inf_{x\in\mathcal{X}} w^\top f(x)$ for every $z \in \mathbb{R}$. But $\inf_{x\in\mathcal{X}} w_z^\top f(x)$ is finite and since $d \in \operatorname{int} C$, $w^\top d > 0$. Hence there exists sufficiently large $\bar{z}$ such that $\inf_{x\in\mathcal{X}} w^\top f(x) < w^\top v + \bar{z}w^\top d$. Therefore, there exists a feasible solution $(\bar{x}, \bar{z})$ where $\bar{x} \in \mathcal{X}$ satisfies $f(\bar{x}) \leq_C v + \bar{z}d$.
Now, we show that for any feasible solution $(x, z) \in \mathbb{R}^{n+1}$, z is bounded below. As $\mathcal{X}$ is compact, (P) is bounded. There exists $a \in \mathbb{R}^p$ such that $\mathcal{P} \subseteq \{a\} + C$. Therefore, for any feasible (x, z), by Proposition 4.2.4, $v + zd \in \{a\} + C$ holds. This implies that for all $w \in C^+$ we have $w^\top(v + zd - a) \geq 0$. Moreover, since $w^\top d > 0$, it is true that $z \geq \frac{w^\top a - w^\top v}{w^\top d}$ for all $w \in C^+$. Thus, $z \geq \sup_{w\in C^+} \frac{w^\top a - w^\top v}{w^\top d}$.

Since there exists a feasible solution $(\bar{x}, \bar{z}) \in \mathbb{R}^{n+1}$, we obtain $\tilde{z} := \sup_{w\in C^+} \frac{w^\top a - w^\top v}{w^\top d} \neq \infty$. Then, $(\mathrm{PS}(v, d))$ can be written equivalently as:

$$\begin{aligned} \text{minimize } & z \\ \text{subject to } & f(x) \leq_C v + zd \\ & x \in \mathcal{X} \\ & \tilde{z} \leq z \leq \bar{z}. \end{aligned}$$

The feasible region of this problem is compact, hence it has an optimal solution, which is also optimal for $(\mathrm{PS}(v, d))$. □

Proposition 4.2.7. *Suppose Assumption 4.0.1 holds. Let $v \notin \mathcal{P}$, $y \in \mathcal{P}$ and $d = y - v$. Then, $(\mathrm{PS}(v, d))$ has an optimal solution.*

Proof. Let $z = 1$. Then, $v + d = v + (y - v) = y \in \mathcal{P} = f(\bar{\mathcal{X}}) + C$ from Remark 4.1.4 Hence, there exists $\bar{x} \in \mathcal{X}$ such that $(\bar{x}, 1)$ is feasible for $(\mathrm{PS}(v, d))$. We now show that for any feasible (x, z), $z \geq 0$ holds. Assume for contradiction that there exist (x, z) that is feasible for $(\mathrm{PS}(v, d))$ such that $z < 0$. We have $v + z(y - v) = (1 - z)v + zy \in \mathcal{P}$. As $z < 0$, $\frac{1}{1-z} \in (0, 1)$. Moreover, since $\mathcal{P}$ is convex, $(1 - z)v + zy \in \mathcal{P}$ and $y \in \mathcal{P}$, we have

$$\frac{1}{1-z}\left((1 - z)v + zy\right) + \left(1 - \frac{1}{1-z}\right) y = v \in \mathcal{P},$$

which is a contradiction. We conclude that for all feasible (x, z), $z \geq 0$ holds. As $(\bar{x}, 1)$ is a feasible solution, we can add $z \leq 1$ as a constraint to $(\mathrm{PS}(v, d))$. Then, (PS(v, d)) with $d = y - v$ is equivalent to

$$\begin{aligned} \text{minimize } & z \\ \text{subject to } & f(x) \leq_C v + zd \\ & z \in [0,1] \\ & x \in \mathcal{X}. \end{aligned}$$

The proof follows as the feasible region of this problem is compact. By Weierstrass theorem, this problem has an optimal solution, which is also optimal for (PS(v, d)). □

The Lagrange dual problem of PS(v, d) can be written as follows ([9]):

$$\begin{aligned} \text{maximize } & \inf_{x \in \mathcal{X}} \{f(x)^\top w\} - v^\top w \\ \text{subject to } & w^\top d = 1 \\ & w \in C^+. \end{aligned}$$

Moreover, it has been shown in [9] that using the primal-dual solution pair, it is possible to find a supporting hyperplane to the upper image.

Proposition 4.2.8 ([9, Proposition 4.7]). *Let $v \in \mathbb{R}^p$, and (x^*, z^*), (w^*) be the optimal solutions for* (PS(v, d)) *and its dual, respectively. Then $H := \{y \in \mathbb{R}^p | \ (w^*)^\top y = (w^*)^\top v + z^*\}$ is a supporting hyperplane for $\mathcal{P}$ at $y^* = v + z^* d$ and $\mathcal{H} = \{y \in \mathbb{R}^p | \ (w^*)^\top y \geq (w^*)^\top v + z^*\}$ contains $\mathcal{P}$.*

Chapter 5

The Algorithm and Variants

In this chapter, we provide an outer approximation algorithm for solving CVOPs and introduce several variants of it. In Section 5.1, we provide the algorithm and prove that it works correctly. Next, in Section 5.2, we list the test examples that are used in computational studies. In Section 5.3, we introduce different direction selection rules and conduct a preliminary computational study to compare these rules. In Section 5.4, we introduce different vertex selection rules. Finally, we discuss relevant algorithms from the literature in Section 5.5.

5.1 The Algorithm

In this section, we describe the main structure of the proposed algorithm. As mentioned before, it is an outer approximation algorithm. It starts by finding an outer approximation to the upper image and update the approximation in each iteration by solving scalarizations. It stops when the approximation is fine enough.

The details of the algorithm can be explained as follows. It starts by solving (WS(z^i)) for all $i = 1, \ldots, l$ where z^i's are the generating vectors of C^+. Optimal solutions $\{x^1, \ldots, x^l\}$ of these models form the initial set of weak minimizers $\bar{\mathcal{X}}$.

Then the initial outer approximation P^0 of $\mathcal{P}$ is set as $P^0 = \bigcap_{i\in\{1,\dots,l\}} \mathcal{H}^i$ (lines 2, 3 of Algorithm 1), where $\mathcal{H}^i = \{y \in \mathbb{R}^q | (z^i)^\top y \geq (z^i)^\top f(x^i)\}$. It is not difficult to see that $\mathcal{H}^i \supseteq \mathcal{P}$ for all i, hence $P^0 \supseteq \mathcal{P}$.

In k^{th} iteration, the vertices V^k of the current outer approximation is considered. A vertex $v \in V^k \setminus V_{used}$ is selected, where V_{used} is the set of selected vertices in the previous iterations and it is initialized as empty set. Later we will discuss different vertex selection methods. Some of these return $V_{info} \neq \emptyset$, which stores the triples (v, y^v, z^v) for the vertices $v \in V^k \setminus V_{used}$, where $y^v \in \mathcal{P}$ and $z^v = \|y^v - v\|$. In this case, an upper bound for the Hausdorff distance between the current approximation and the upper image, namely $\hat{h} = \max_{v\in V^k \setminus V_{used}} z^v$ can be computed (line 8), see also Proposition 3.0.1. If $\hat{h} \leq \epsilon$, the algorithm terminates by letting $V^k = \emptyset$ and *solve* $= 0$. Otherwise, it remains *solve* $= 1$ (lines 8-10). $V^k_{info} = \emptyset$ means that the vertex selection method does not store any information about the current vertices.

If $V^k_{info} \neq \emptyset$ but *solve* $= 1$ or $V^k_{info} = \emptyset$, then the Pascoletti-Serafini scalarization (PS(v, d)) is solved in order to find a weak minimizer x^v, see Proposition 4.2.3. If the direction parameter is not fixed from the beginning of the algorithm, then it has to be computed first. Later, we discuss different ways of computing the direction parameter. The selected v is added to V_{used} and the corresponding weak minimizer x^v is added to set $\bar{\mathcal{X}}^k$ (lines 13-16). If the current vertex is close enough to the upper image, then the algorithm checks another vertex from $V^k \setminus V_{used}$. Otherwise, the current outer approximation is updated by intersecting it with the supporting halfspace $\mathcal{H}$ of $\mathcal{P}$ at $f(x^v)$, see Proposition 4.2.8. The vertices of the updated outer approximation is computed by solving a vertex enumeration problem (lines 18-21). The algorithm terminates if all the vertices of the current outer approximation are in ϵ distance to the upper image and returns a set of weak minimizers $\bar{\mathcal{X}}$.

Remark 5.1.1. *Note that the weak minimizers can also be added only if* $z^v \leq \epsilon$ *instead of adding all weak minimizers to the solution set to obtain a coarser set of solutions.*

Algorithm 1 Primal Approximation Algorithm for (P)

```
1: $V_{used} = \emptyset$, $\bar{\mathcal{X}}^0 = \emptyset$, $k = 0$, $solve = 1$
2: For $i = 1, \dots, l$: Solve (WS($z^i$)) to find $x^i$ , $\bar{\mathcal{X}}^0 \leftarrow \bar{\mathcal{X}}^0 \cup \{x^i\}$ and let
   $\mathcal{H}^i = \{y \in \mathbb{R}^q | {z^i}^\top y \geq {z^i}^\top f(x_i)\}$.
3: Let $P^0 := \bigcap_{i \in \{1,\dots,l\}} H^i$, compute the initial set of vertices $V^0$ of $P^0$.
4: Fix rules for selecting
   - the order to pick the vertices from $V$ and
   - the direction parameter $d$.
5: while $V^k \setminus V_{used} \neq \emptyset$ do
6:     $[v, V^k_{info}]$ ←SelectVertex($V^k, V_{used}$)
7:     if $V^k_{info} \neq \emptyset$ then
8:         if $\hat{h} = \max_{v \in V^k \setminus V_{used}} z^v \leq \epsilon$ then
9:             $V^k = \emptyset$, $solve = 0$;
10:        end if
11:    end if
12:    if $V^k_{info} = \emptyset$ or $solve = 1$ then
13:        Compute $d^v$ (if not fixed) such that $d^v \in \operatorname{int} C$ and $\|d^v\| = 1$.
14:        Solve (PS($v, d^v$)). Let $(x^v, z^v)$ be an optimal solution.
15:        $V_{used} \leftarrow V_{used} \cup \{v\}$.
16:        $\bar{\mathcal{X}}^k \leftarrow \bar{\mathcal{X}}^k \cup \{x^v\}$.
17:        if $z^v > \epsilon$ then
18:            Find supporting halfspace $\mathcal{H}$ of $\mathcal{P}$ at $y^v = v + z^v d^v$ (see Proposition 4.2.8).
19:            $P^{k+1} \leftarrow P^k \cap \mathcal{H}$, $\bar{\mathcal{X}}^{k+1} \leftarrow \bar{\mathcal{X}}^k$.
20:            Compute the set of vertices $V^{k+1}$ of $P^{k+1}$.
21:            $k \leftarrow k + 1$.
22:        end if
23:    end if
24: end while
25: return $\bar{\mathcal{X}}^k$.
```

Later, we will discuss different rules for selecting the direction parameter in

Section 5.3 and different vertex selection rules in Section 5.4. The following proposition holds true for any selection rule.

Proposition 5.1.2. *Suppose Assumption 4.0.1 holds. When terminates, Algorithm 1 returns a finite weak ϵ-solution.*

Proof. There exists optimal solutions to (WS(z^i)) for all $i \in \{1, \dots, l\}$ by Assumption 4.0.1. Initial polyhedron P^0 found in line 3 has at least one vertex as C is pointed and (P) is bounded, see for instance [14, Corrolary 18.5.3]. Hence, the set V^0 (consequently $V^0 \setminus V_{used}$) is nonempty. Note that the initial solution set $\bar{\mathcal{X}}^0$ contains weak minimizers by Proposition 4.2.1. At iteration k, SelectVertex() returns $v \in V^k \setminus V_{used}$ and V^k_{info}. First consider the case $V^k_{info} = \emptyset$. As $d^v \in \operatorname{int} C$ is ensured, by Proposition 4.2.6, there exists a solution (x^k, z^k) to (PS(v, d^v)). By Proposition 4.2.3, x^k is a weak minimizer. Hence, for any iteration k, $\bar{\mathcal{X}}^k$ is finite and contains only the weak minimizers. By Proposition 4.2.8, $\mathcal{H}$ computed in line 18 is a supporting halfspace. Then, P^{k+1} in line 19 has at least one vertex and satisfies $P^{k+1} \supseteq \mathcal{P}$.

The algorithm terminates when $V^k \setminus V_{used} = \emptyset$. Assume this is the case after K iterations. This suggests that each $v \in V^K$ is also an element of V_{used} and $z^v \leq \epsilon$ for all $v \in V^K$ at termination. To show that $\bar{\mathcal{X}}^K$ is a weak ϵ-solution of (P), i.e., $\mathcal{P} \subseteq \operatorname{conv} f(\bar{\mathcal{X}}^K) + C + B(0, \epsilon)$, it is sufficient to show that

$$P^K \subseteq \operatorname{conv} f(\bar{\mathcal{X}}^K) + C + B(0, \epsilon). \tag{5.1}$$

Note that the recession cone of P^k is C [11, Lemma 5.2]. Hence, we have $P^k = \operatorname{conv} V^K + C$. Then, it is enough to show

$$\operatorname{conv} V^K + C \subseteq \operatorname{conv} f(\bar{\mathcal{X}}^K) + C + B(0, \epsilon). \tag{5.2}$$

Let $\sum_{v \in V^K} \lambda^v v + \bar{c}$ be arbitrary in $\operatorname{conv} V^K + C$, where $\sum_{v \in V^K} \lambda^v = 1$, $\lambda^v \geq 0$ for all $v \in V^K$ and $\bar{c} \in C$. For $v \in V^K$, since (z^v, x^v) is an optimal solution of (PS(v, d^v)), there exist $c^v \in C$ such that $v + d^v z^v = f(x^v) + c^v$. Note that for all

$v \in V^K$, we have $z^v \leq \epsilon$ and $x^v \in \bar{\mathcal{X}}^K$. Then, we have

$$\begin{aligned}\sum_{v \in V^K} \lambda^v v + \bar{c} &= \sum_{v \in V^K} \lambda^v (f(x^v) + c^v - d^v z^v) + \bar{c} \\ &= \sum_{v \in V^K} \lambda^v f(x^v) + \sum_{v \in V^K} \lambda^v c^v + \bar{c} - \sum_{v \in V^K} \lambda^v d^v z^v\end{aligned}$$

Clearly, $\sum_{v \in V^k} \lambda^v f(x^v) \in \operatorname{conv} f(\bar{\mathcal{X}}^K)$ and $\sum_{v \in V^K} \lambda^v c^v + \bar{c} \in C$. Moreover, as $z^v \leq \epsilon$ and $\|d^v\| = 1$, $z^v d^v \in B(0, z^v) \subseteq B(0, \epsilon)$. This implies $\sum_{v \in V^K} \lambda^v z^v d^v \in B(0, \epsilon)$. Hence,

$$\sum_{i=1}^{k} \lambda^v v + \bar{c} \in \operatorname{conv} f(\bar{\mathcal{X}}^K) + C + B(0, \epsilon).$$

For the vertex selection rules that give $V^k_{info} \neq \emptyset$, an alternative stopping criteria is used in line 9. The algorithm terminates if $\hat{h} = \max_{v \in V^k \setminus V^k_{used}} z^v \leq \epsilon$. Assume this is the case after K iterations, that is $\max_{v \in V^k \setminus V_{used}} z^v \leq \epsilon$. Note that if there exists a vertex v in $V^K \cap V_{used}$, then $z^v \leq \epsilon$ has to be satisfied by the structure of the algorithm. Hence, for the vertices V^K of P^K, we have $z^v \leq \hat{h} \leq \epsilon$. The similar steps for the previous case can be applied to show that the algorithm returns a finite weak ϵ-solution when it terminates.

□

Depending on the selection of the parameters v, d^v of the scalarization model, it is possible to generate many variants of Algorithm 1. Indeed, the algorithms proposed in [7, 9, 8, 10] fit into this framework. The (primal) algorithm in [9] is in the form of Algorithm 1, where d^v is fixed throughout the algorithm. Similarly, the algorithm in [8] is of this type, where d^v changes in every iteration depending on the current vertex v and a fixed point $\hat{p}$ in the interior of the upper image by setting $d^v = \hat{p} - v$. There is no fixed rule for selecting the vertices in both algorithms; v is selected from the list of vertices arbitrarily. Note that the algorithm in [8] differs from Algorithm 1 in the sense that it does not use the dual solution of the scalarization problem to find the supporting hyperplane, instead uses the derivatives of the objective functions. In order to have a fair comparison, for our computational tests in Chapter 5, we use Proposition 4.2.8 to generate a supporting hyperplane to the upper image also for this algorithm.

The algorithms proposed in [7] and [10] select the vertices by solving repetitive models at each iteration. While doing this, [10] solves quadratic models that benefits from the inner approximation, and [7] solves gauge-based scalarizations.

There is also a recent algorithm proposed in [11], which fits into the general framework of Algorithm 1. The main difference is that instead of solving Pascoletti-Serrafini models, it solves norm-minimizing scalarizations in each iteration.

5.2 Test Examples

Before we proceed with the different variants of the algorithm, we provide in this section some numerical examples that will be used throughout this thesis. In all of these examples, the ordering cone is taken as the usual nonnegative cone, $\mathbb{R}^p_+$.

Example 1. Consider the so-called unit ball example:

$$\begin{aligned} \text{minimize} \quad & x \\ \text{subject to} \quad & \|x - e\| \leq 1 \\ & x \geq 0, x \in \mathbb{R}^n \end{aligned}$$

Example 2. Consider the following biobjective example

$$\begin{aligned} \text{minimize} \quad & \left(x,\ \frac{1}{x}\right)^\top \\ \text{subject to} \quad & x \geq 0, x \in \mathbb{R} \end{aligned}$$

This example is bounded in the sense that the upper image is included in $y^I + \mathbb{R}^2_+$ where $y^I = 0 \in \mathbb{R}^2$. In theory, the weighted sum scalarizations for the initialization step do not have a solution. However, due to the precision levels of the solvers, the weighted sum scalarizations can return solutions. We manually compute the initial outer approximation as $P^0 = \{0\} + \mathbb{R}^2_+$, but we also include the initial solutions found by the scalarization, because they are feasible by the problem structure.

Example 3. The following set of examples are taken from [10].

$$\begin{aligned} &\text{minimize } (x_1\ x_2\ x_3)^\top \\ &\text{subject to } \left(\frac{x_1-1}{1}\right)^2 + \left(\frac{x_2-1}{a}\right)^2 + \left(\frac{x_3-1}{5}\right)^2 \leq 1 \end{aligned}$$

for $a \in \{5, 7, 10, 20\}$.

Example 4. We generate examples with $p = 4$ using a similar structure as in Example 3.

$$\begin{aligned} &\text{minimize } (x_1\ x_2\ x_3\ x_4)^\top \\ &\text{subject to } \left(\frac{x_1-1}{1}\right)^2 + \left(\frac{x_2-1}{a}\right)^2 + \left(\frac{x_3-1}{5}\right)^2 + \left(\frac{x_4-1}{1}\right)^2 \leq 1 \end{aligned}$$

for $a \in \{5, 7, 10\}$.

Example 5. We consider the linear problems given by

$$\begin{aligned} \text{minimize} \quad & P^\top x \\ \text{subject to} \quad & A^\top x \leq b, \end{aligned}$$

where $P \in \mathbb{R}^{p\times n}$, $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$.

We generate random instances of this type, where each component of A and P is generated according to independent normal distributions with mean 0, and variance 100; each component of b is generated according to a uniform distribution between 0 and 10. Then, each component of A, P and b is rounded to the nearest integer for computational simplicity.

5.3 Direction Selection Rules

In this section, we discuss some direction selection rules. We start by recalling the algorithms from the literature. Throughout the Primal Approximation Algorithm in [9], a fixed direction parameter is used within the convex optimization problem that is solved in each iteration, namely d in (PS(v, d)) is fixed. While finding a point on $\operatorname{bd}\mathcal{P}$, the distance is calculated from vertex v following a fixed direction. On the other hand, for the algorithm in [8], instead of fixing the direction parameter, a point $\hat{p} \in \mathcal{P}$ is fixed and in each iteration, d^v is taken as $\hat{p} - v$.

Before proceeding with the proposed direction selection rules and the preliminary numerical tests, we use Example 1 and Example 2 to show that the selection of the fixed d parameter in [9] and the fixed point $\hat{p}$ in [8] affect the performance of these algorithms significantly.

Table 5.1: Results for different d and $\hat{p}$ values

	Example 1 ($\epsilon = 0.005$)								**Example 2** ($\epsilon = 0.05$)					
	d [9]				$\hat{p}$ [8]				d [9]			$\hat{p}$ [8]		
	$\begin{pmatrix}1\\1\end{pmatrix}$	$\begin{pmatrix}0.1\\1\end{pmatrix}$	$\begin{pmatrix}0.01\\1\end{pmatrix}$	$\begin{pmatrix}0.001\\1\end{pmatrix}$	$\begin{pmatrix}1\\1\end{pmatrix}$	$\begin{pmatrix}0.1\\1\end{pmatrix}$	$\begin{pmatrix}0.01\\1\end{pmatrix}$	$\begin{pmatrix}0.001\\1\end{pmatrix}$	$\begin{pmatrix}1\\1\end{pmatrix}$	$\begin{pmatrix}0.1\\1\end{pmatrix}$	$\begin{pmatrix}0.01\\1\end{pmatrix}$	$\begin{pmatrix}100\\100\end{pmatrix}$	$\begin{pmatrix}10\\1000\end{pmatrix}$	$\begin{pmatrix}10\\10000\end{pmatrix}$
SC	29	27	35	41	17	31	39	47	31	56	304	30	178	741
Iteration	14	13	17	20	8	15	19	23	12	22	38	12	35	63
Time	10.10	9.94	13.34	14.62	6.27	11.06	13.97	23.76	13.60	18.19	93.81	10.52	61.12	226.63

We employ the algorithms from [9] and [8] with different values of d and $\hat{p}$ on Example 1 for $p = 2$ and Example 2. The results are summarized in Table 5.1, in which we report the number of scalarization models (SC), the number of iterations (Iteration), and the solution time (Time) in seconds.

As can be seen from Table 5.1, the choices of the fixed direction d in [9] and the fixed point $\hat{p}$ in [8] affect the number of models, number of iterations, hence the required CPU time to solve the same problem. These choices clearly affect the efficiency of the algorithm.

Motivating from these results, we propose different rules to choose the direction parameter to be used in Algorithm 1. Among these, the second approach explained in Section 5.3.2 is designed only for MOPs.

5.3.1 Adjacent Vertices Approach (Adj)

In order to choose a direction parameter for a vertex v of the current outer approximation, it is possible to use the coordinates of the adjacent vertices it. Note that in line 20 of Algorithm 1, we solve a vertex enumeration problem. The toolbox introduced by Löhne and Weissing [15, 16], namely *bensolve tools* uses the double description method in order to solve the vertex enumeration problem. This method allows us to find the adjacent vertices of a vertex of the current outer approximation. After the adjacent vertices are found, the normal direction of an hyperplane passing through these adjacent vertices is computed.

Let the adjacent vertices of the vertex v be $v^1_{adj}, \dots, v^s_{adj}$ and form matrix A as $A = [v^1_{adj}, \dots, v^s_{adj}] \in \mathbb{R}^{p \times s}$. It is possible that a vertex may have more adjacent vertices than required to define a hyperplane, i.e., $s > p$. In that case, we choose p among s adjacent vertices and form the matrix $A \in \mathbb{R}^{p \times p}$ such that its columns are linearly independent. Let the plane passing through the selected adjacent vertices be $K =: \{x \in \mathbb{R}^p | \; x^\top c = b\}$ where c is the normal vector. Note that $A^\top c = b$ holds. Since the columns of A are linearly independent, this system has a unique solution c. Clearly, K can have two unit normal vectors $d = \frac{c}{\|c\|}$ and $d = \frac{-c}{\|c\|}$. Note that if $d \in \operatorname{int} C$ we obtain weak minimizers by solving (PS(v, d)), see Proposition 4.2.6. In order to guarantee this, we first check if the two candidate unit normal vectors satisfies this. For this purpose, we use a predetermined direction $d \in \operatorname{int} C$ where $\|d\| = 1$. For the numerical examples we set $d = \frac{\sum_{i=1}^{l} z^i}{\|\sum_{i=1}^{l} z^i\|}$, where $\{z^1, \dots, z^l\}$ are the generating vectors of C^+ for the numerical examples.

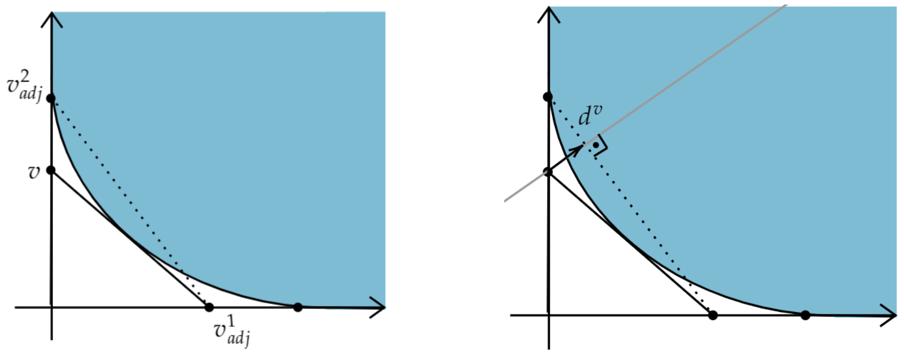
Note that the adjacency list of a vertex may contain extreme directions of P^k, that is, for some vertices, in addition to the adjacent vertices, we may also have adjacent extreme directions. If for a vertex v, we have an adjacent extreme direction z, then it is known that the line segment $\{v + rz \in \mathbb{R}^p | \, r \geq 0\}$ is a face of P^k. In these cases, we construct an artificial adjacent vertex by moving the current vertex along the adjacent extreme direction, that is we take $v_{adj} = v + z$. The pseudocode of this rule is given in Procedure 2, and an illustration for $p = 2$ is given in Figure 5.1.

Procedure 2 ChooseDirection(v,P)

```
 1: Let $A = [v_{adj}^1 \; v_{adj}^2 \; \cdots \; v_{adj}^s]$ be the vertices adjacent to $v$
 2: if $s > p$ then
 3:     Choose $p$ linearly independent columns of $A$. Let this submatrix be $A$.
 4: end if
 5: Compute $d = \frac{A^{-\top}e}{\|A^{-\top}e\|}$
 6: if $d \in \operatorname{int} C$ then
 7:     return $d$
 8: else if $-d \in \operatorname{int} C$ then
 9:     return $-d$
10: else
11:     return A predetermined $d \in \operatorname{int} C$ with $\|d\| = 1$.
12: end if
```

For an example where Procedure 2 executes line 11, see Appendix A.2.1. For $p = 2$ and $C = \mathbb{R}^2_+$, the direction found in Line 7 or 9 of Procedure 2 is in the interior of $\mathbb{R}^2_+$, see Appendix A.2.2.

Figure 5.1: Direction selection based on (Adj)



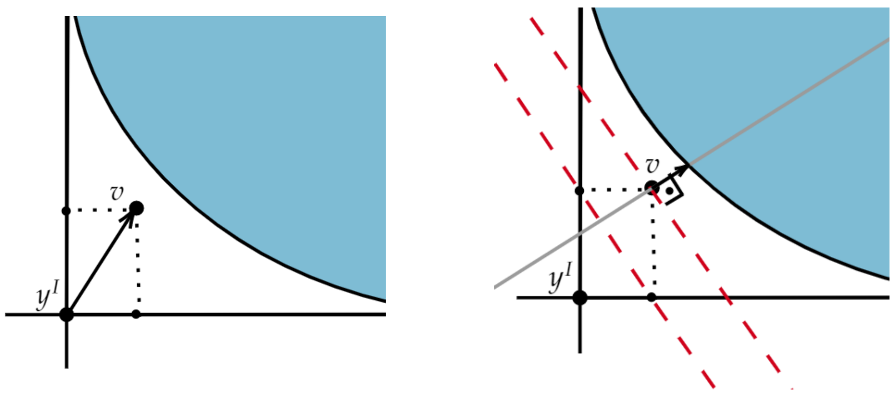
5.3.2 Ideal Point Based Approach (IP)

For this approach, we assume that the ordering cone is the positive orthant, hence the ideal point y^I is well defined. For a vertex v of the current outer approximation, we consider the vector $v - y^I$. Note that for the initial iteration, we have $v - y^I = 0$, as y^I is the only vertex of the initial outer approximation. In the subsequent iterations, we obtain $v - y^I \in \mathbb{R}^p_+$ since $P^k \subseteq P^0$ for any k throughout the algorithm.

The direction is set as $d^v = \frac{\hat{d}}{\|\hat{d}\|}$ where $\hat{d}$ be such that $\hat{d}_i = \frac{1}{y^I_i - v_i}$ for $i \in \{1, \dots, p\}$. Geometrically, this corresponds to considering points $(v_i - y^I_i)e_i$ for each $i \in \{1, \dots, p\}$, and taking the normal direction of the hyperplane passing through them. See Figure 5.2 that shows an illustration of this method.

In order to ensure $d^v \in \mathbb{R}^p_{++}$, that is, $d^v_i > 0$ for all $i \in \{1, \dots, p\}$, we set $\hat{d}^v_i = \frac{1}{y^I_i - v_i + \bar{\epsilon}} > 0$ for all i which implies $d^v \in \mathbb{R}^p_{++}$. For the numerical examples, we take $\bar{\epsilon} = 10^{-5}$.

Figure 5.2: Direction Selection Based on (IP)



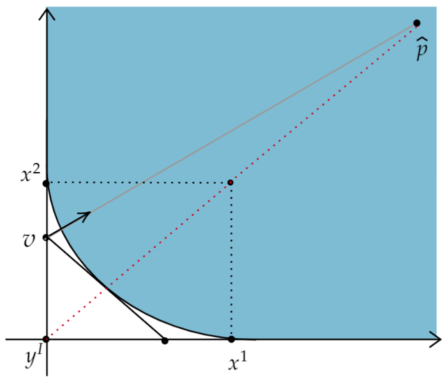
5.3.3 Fixed Point Approach (FP)

For this approach we use the idea presented in [8] and use a fixed point $\hat{p} \in \mathcal{P}$ in order to determine the direction parameter. For the examples in Section 5.2, instead of choosing an arbitrary fixed point, we fix $\hat{p}$ such that

$\hat{p}_i := 2\max\{f_i(x^1), \dots, f_i(x^p)\} - v_i,\ i = 1, \dots, p$ where $x_i,\ i \in \{1, \dots, p\}$ are optimal solutions of $(\mathrm{WS}(z^i))$ found in the initialization step and v is a vertex of the initial polyhedron P^0. Then, the changing direction is set as $d = \frac{\hat{p}-v}{\|\hat{p}-v\|}$. Note that for a MOP, we have $v = y^I$. See Figure 5.3 for an illustration for $p = 2$.

By Proposition 4.2.7, there is an optimal solution to $\mathrm{PS}(v, d)$ under this direction selection rule. However, for the computational tests, we ensure $d \in \operatorname{int} C$ for computational simplicity and to reduce the numerical issues. Accordingly, if $d \notin \operatorname{int} C$, then we use a predetermined direction d as explained in Section 5.3.1.

Figure 5.3: Direction selection based on (FP)



5.3.4 Preliminary Computational Study

To compare the direction parameter selection rules, we conduct a preliminary computational study. Since vertex selection will also play an important role in the efficiency of the algorithm, we employ different "simple" vertex selection rules (VSRs) for this preliminary computational study. Below we summarize these vertex selection rules briefly:

- **Closest to the Ideal Point (Id)**

This approach assumes that the ordering cone is $\mathbb{R}^p_+$ and the ideal point y^I is well-defined. The rule is to choose $v \in V^k \setminus V_{used}$ such that $\left\|v - y^I\right\|$ is minimum, see Procedure 3.

Procedure 3 SelectVertex(V^k,V_{used},y^I)

```
1: $\mathcal{V}^k = \emptyset$
2: for all $v \in V^k \setminus V_{used}$ do
3:     Calculate $dist_v = \left\|v - y^I\right\|$
4:     $\mathcal{V}^k \leftarrow \mathcal{V}^k \cup \{(v, dist_v)\}$
5: end for
6: Pick $(v^*, dist^*_v) \in \mathcal{V}^k$ such that $dist^*_v = \min_{(v,dist_v)\in\mathcal{V}^k} dist_v$
7: return $v^*$
```

- **Closest to an Inner Point (In)**

  For each $v \in V^k \setminus V_{used}$ the Euclidean distance $\|\hat{p} - v\|$ is calculated and the vertex with the greatest distance is chosen to be the reference point, see Procedure 4.

Procedure 4 SelectVertex(V^k,V_{used},$\hat{p}$)

```
1: $\mathcal{V}^k = \emptyset$
2: for all $v \in V^k \setminus V_{used}$ do
3:     Calculate $dist_v = \|\hat{p} - v\|$
4:     $\mathcal{V}^k \leftarrow \mathcal{V}^k \cup \{(v, dist_v)\}$
5: end for
6: Pick $(v^*, dist^*_v) \in \mathcal{V}^k$ such that $dist^*_v = \max_{(v,dist_v)\in\mathcal{V}^k} dist_v$
7: return $v^*$
```

- **Random Choice (R)**

  The vertex is chosen randomly among $V^k \setminus V_{used}$, using discrete uniform distribution over $V^k \setminus V_{used}$. For numerical examples provided below, we run the same example five times for this vertex selection rule.

For the test examples in Section 5.2, we run the algorithm for different direction selection rules (Fixed, Adj, IP, FP) and vertex selection rules (Id, R, In) and report the total number of scalarization models solved (SC) as well as the total CPU time required, see Tables 5.2-5.6. For the fixed direction variant (Fixed), d is set to $d = \frac{e}{\|e\|}$.

For some of the instances, *bensolve tools* was unable to perform vertex enumeration due to numerical issues. These are indicated by (-) in the tables.

Table 5.2: Computational Results for Example 1

	Direction Selection Rule															
	p=3 , ϵ =0.005								p=4, ϵ = 0.05							
	Fixed		Adj		IP		FP		Fixed		Adj		IP		FP	
VSR	SC	Time	SC	Time	SC	Time	SC	Time	SC	Time	SC	Time	SC	Time	SC	Time
Id	488	134.73	425	116.76	627	169.9	461	126.79	900	275.15	1099	328.62	600	177.79	682	205.87
R	461.2	127.6	392.2	107.55	388	105.51	407.8	111.29	460	153.96	460.6	140.05	420	126.76	416	128.83
In	502	134.4	420	111.77	457	118.89	436	114.06	755	232.26	605	183.87	-	-	-	-

Table 5.3: Computational Results for Example 2, $\epsilon = 0.005$

	Direction Selection Rule							
	Fixed		Adj		IP		FP	
VSR	SC	Time	SC	Time	SC	Time	SC	Time
Id	154	35.38	134	31.11	141	31.83	283	63.99
R	127.8	27.64	120.8	26.54	113	24.49	223.6	48.4
In	222	48.48	165	35.26	131	28.74	271	58.93

Table 5.4: Computational Results for Example 3, $\epsilon = 0.05$

	Direction Selection Rule															
	a = 5								a = 7							
	Fixed		Adj		IP		FP		Fixed		Adj		IP		FP	
VSR	SC	Time	SC	Time	SC	Time	SC	Time	SC	Time	SC	Time	SC	Time	SC	Time
Id	142	38.28	155	41.79	114	31.45	187	50.1	150	39.71	134	35.55	160	42.25	291	77.27
R	125.8	33.56	102.4	27.34	109	29.03	173.4	46.6	123.2	32.77	111.4	29.85	115.4	30.55	204.6	54.62
In	120	31.73	116	30.77	124	32.63	180	47.67	146	38.97	113	30.77	120	31.98	224	60.08
	a = 10								a = 20							
	Fixed		Adj		IP		FP		Fixed		Adj		IP		FP	
VSR	SC	Time	SC	Time	SC	Time	SC	Time	SC	Time	SC	Time	SC	Time	SC	Time
Id	150	39.95	123	33.7	130	34.69	273	74.12	204	55.04	141	38.26	148	39.87	517	144.02
R	138.2	36.78	118.8	31.64	108.2	28.66	250.6	67.15	162	43.81	128.6	34.86	147.2	39.43	472.8	130.1
In	156	41.21	136	35.58	129	34.25	290	76.92	153	41.56	130	35.09	144	38.99	666	183.5

Table 5.5: Computational Results for Example 4, $\epsilon = 0.05$

	Direction Selection Rule															
	a = 5								a = 7							
	Fixed		Adj		IP		FP		Fixed		Adj		IP		FP	
VSR	SC	Time	SC	Time	SC	Time	SC	Time	SC	Time	SC	Time	SC	Time	SC	Time
Id	5529	2085.9	2497	630.53	-	-	7218	2806.85	8153	2538.08	-	-	1973	646.46	7099	2938.67
R	1185	424.59	1107.4	379.02	-	-	-	-	1247.4	450.86	1139	397.09	-	-	3093.6	1567.57
In	2401	800.11	1584	526.82	-	-	5742	2203.63	2956	1012.39	4969	1554.33	3452	1013.43	6327	2771.90

	a = 10							
	Fixed		Adj		IP		FP	
VSR	SC	Time	SC	Time	SC	Time	SC	Time
Id	3292	1120.34	2807	880.45	3050	937.37	-	-
R	1359.6	498.34	1150.6	396.60	-	-	-	-
In	-	-	2098	710.81	2106	683.75	11876	5970.35

Table 5.6: Computational Results for Example 5, $\epsilon = 10^{-8}$

	Direction Selection Rule															
	d = 2								d = 3							
	Fixed		Adj		IP		FP		Fixed		Adj		IP		FP	
VSR	SC	Time	SC	Time	SC	Time	SC	Time	SC	Time	SC	Time	SC	Time	SC	Time
Id	73.75	13.93	73.3	13.86	70.95	13.41	72.75	13.73	308.45	54.64	301.25	53.55	292.5	52.04	357	62.89
R	71.55	13.74	72.45	13.39	70.75	13.47	71.3	13.46	192.05	34.97	194.8	35.5	186.2	33.91	200.45	36.49
In	75.2	14.12	72.15	13.64	71.1	13.44	73.2	13.87	337	59.45	283.95	50.56	294.15	51.99	327.6	57.59

The results in Table 5.6 display the average number of scalarizations and CPU times of 20 randomly generated problem instances.

We observe that while there is no direction method that consistently outperforms the others for Examples 1 and 5, (Fixed) is always outperformed by some direction selection rule for the same vertex selection method, see Table 5.2 and 5.6. The improvements made by the direction selection rules (compared to (Fixed)) are more significant for the Example 3 and Example 4 compared to Example 1 and Example 5, especially when a takes larger values. In these examples, (Adj) and (IP) tend to decrease the CPU times and number of models solved besides some exceptions, see Table 5.4 and Table 5.5. The variants that use (FP) as a direction choice lag behind others for Example 3 and Example 4. The negative impact of direction rule (FP) is clearer when the image of the feasible region is dilated along x_2 axis, that is when a is larger. In Example 2, (Adj) and (IP) consistently decrease the number of models to be solved for each vertex selection method, see Table 5.3. The direction rule (FP) performs poorly compared to (Fixed), (Adj), and (IP) for the same example. Similar to Example 3 and Example 4, (R-Adj) and (R-IP) are faster than the other variants.

Based on these results, for our future analysis, we fix (Adj) as a direction selection rule. Note that (IP) benefits from the ideal point concept, hence can only be used in the MOP setting; whereas (Adj) can be used also for VOPs.

Among the vertex selection methods, for the same choice of direction rule, (R) performs significantly better in all problems than the vertex selection methods (In) and (Id). The reason that (Id) and (In) decrease the algorithm's performance can be the result of these methods' tendency to choose vertices that are close to each other especially for certain problem instances. On the other hand, the strength of (R) over (Id) and (In) comes from its uniform choice of vertices among the set, which makes it less likely to select vertices closer to the vertex chosen in the previous iteration. However, (R) also has a downside because it may not always provide consistent results. This motivates us to develop different vertex selection rules that are likely to give "better" results.

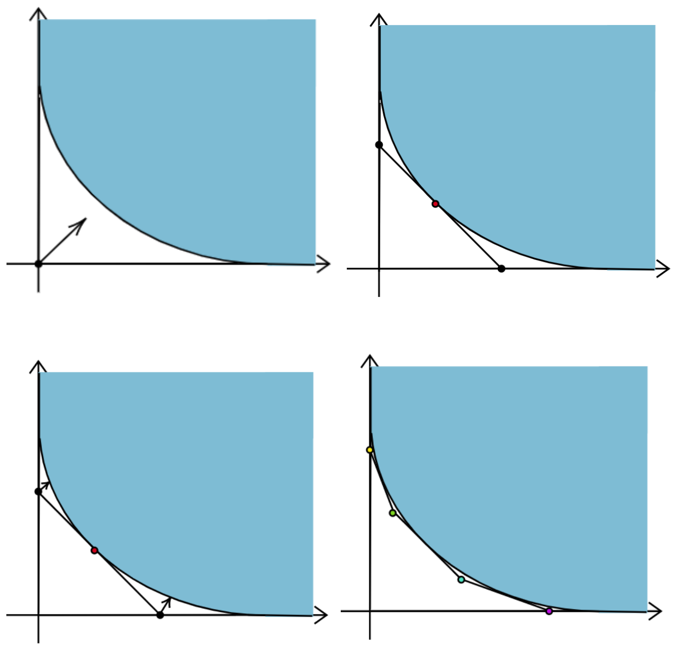
5.4 Vertex Selection Rules

We propose different vertex selection rules that can be used along within Algorithm 1. Note that there are algorithms from the literature, see [10, 7] with different vertex selection rules, which require solving additional optimization models, see also Section 5.5. Our motivation is to propose some vertex selection rules which are less time consuming yet have the potential to work efficiently.

5.4.1 Vertex selection with clusters (C)

This vertex selection rule groups the vertices of the current polyhedron and visits these groups sequentially for vertex selection. The motivation is to ensure that the vertices from different regions of the outer approximation are selected in a balanced fashion. The first step is to fix centers for the clusters. Then, each vertex is assigned to the cluster with the closest center.

Figure 5.4: Selecting the centers for (C)



Procedure 5 SelectCenters(V^0,P^0)

```
1:  for $k = 0 : 1$ do
2:      for all $v \in V^k$ do
3:          Solve PS$(v, d)$. Let $(x^v, z^v)$ be the optimal solution, $\bar{\mathcal{X}} \leftarrow \bar{\mathcal{X}} \cup \{x^v\}$.
4:          Find supporting halfspace $\mathcal{H}$ of $\mathcal{P}$ at $y^v = v + z^v d$, $P^{k+1} \leftarrow P^k \cap H$
5:      end for
6:      Compute the set of vertices $V^{k+1}$ of $P^{k+1}$.
7:  end for
8:  $C = \emptyset$.
9:  for $v \in V^2$ do
10:     Let $c_i \leftarrow v$ and $C \leftarrow C \cup \{c_i\}$.
11: end for
12: return $C$
```

The procedure to select the centers, which is given in Procedure 5, returns a set of clusters C. It first solves (PS(v, d)) models for each vertex of the initial polyhedron P^0. The halfspaces are intersected with the current polyhedron. The procedure is repeated for the vertices of the updated polyhedron P^1. Then, the resulting vertices of P^2 are selected as the centers of the clusters. See Figure 5.4 for the illustration of center selection procedure. The centers are fixed throughout the algorithm and the vertices are assigned to the cluster with the closest center with respect to the Euclidean distance at each iteration.

Then, a vertex is chosen arbitrarily from that cluster. If there is no unexplored vertex assigned to the current cluster, then the algorithm selects the next nonempty cluster. To visit different clusters, we benefit from an index t which is initialized as 0 in the beginning of the algorithm. We increment t by 1 and use mod function to find the remainder of t divided by the total number of clusters, namely $|C|$. Note that $\mod (t, |C|)$ can take values between 0 and $|C| - 1$. Hence, we calculate $\mod (t, |C|) + 1$ to find the index of the cluster. The pseudocode for vertex selection rule (C) is given in Procedure 7.

Procedure 6 SelectVertex(t,V^k,V_{used},C)

```
1: for all $v \in V^k \setminus V_{used}$ do
2:     $Dist_v = \emptyset$
3:     for all $i \in \{1, \cdots, |C|\}$ do
4:         Let $C_i$ be the $i^{th}$ cluster with center $c_i$ and let $C_i = \emptyset$.
5:         $dist_i = \|v - c_i\|$ and $Dist_v \leftarrow Dist_v \cup \{dist_i\}$
6:     end for
7:     Pick $i \in \underset{dist_i \in Dist_v}{\arg\min}\, dist_i$. $C_i \leftarrow C_i \cup \{v\}$.
8: end for
9: $current = \mod(t, |C|) + 1$.
10: while $C_{current} = \emptyset$ do
11:     Let $t = t + 1$ and $current = \mod(t, |C|) + 1$
12: end while
13: Pick an arbitrary $v \in C_{current}$.
14: $t = t + 1$.
15: return $v$, $t$
```

5.4.2 Vertex selection with adjacency information (Adj)

Recall that *bensolve tools* returns the adjacency information for the vertices of the outer approximation. The idea is to use that information to detect "isolated" vertices of the outer approximation. The motivation is to obtain "uniformity" among the vertices of the outer approximation and consequently, the C-minimal points found on the boundary of the upper image. For each vertex v of the current outer approximation, the procedure finds the minimum distance from v to its current neighbors say $dist_v$. Then, it selects the vertex which has the maximum $dist_v$. See Procedure 7 for details.

Procedure 7 SelectVertex(V^k,V_{used})

```
1: for all $v \in V^k \setminus V_{used}$ do
2:     Let $A^v$ be the set of vertices adjacent to $v$.
3:     Let $dist_v = \min_{\tilde{v} \in A^v} \|\tilde{v} - v\|$.
4: end for
5: return  $v^* \in \arg\max_{v \in V^k \setminus V_{used}} dist_v$.
```

5.4.3 Vertex selection using local upper bounds (UB)

For this variant, the motivation is to find local upper bounds for the current set of vertices and then to use those upper bounds in order to select a vertex. In order to compute these upper bounds which are guaranteed to be in $\mathcal{P}$, we adapt an algorithm, namely the p-split algorithm, which is originally designed to solve multiobjective integer programming problems, see for instance [18, 19, 20]. Note that the vertex selection rule that will be described here can only be applied when the problem is a MOP.

To apply this vertex selection rule, some modifications are made at Algorithm 1. In line 1, we additionally fix an upper bound $u = Me$ where M is a sufficiently large number such that $\{u\} - \mathbb{R}^p_+ \supseteq f(\mathcal{X})$ and we initialize the set of upper bounds as $U = \{(u, \emptyset)\}$. Here, $\emptyset$ means that the upper bound u is not defined based on any other point found in the algorithm. Note that the initial upper bound u satisfies that $y^I \leq u$. Through the algorithm, for any vertex v of P^k, it is guaranteed that there exists a local upper bound $\bar{u}$ such that $v \leq \bar{u}$. Indeed, there will be a 'corresponding' local upper bound for each vertex as it will be explained later.

Let v be a vertex for which we solve (PS(v, d)), the corresponding local upper bound for v be u and the (x^v, z^v) be an optimal solution for (PS(v, d)). Using the point $y := v + z^v d$, which is on the boundary of the upper image, p new upper bounds $u^1, \dots, u^p$ are generated as follows: for $j \in \{1, \dots, p\}$, we set $u^j_i = u_i$ for each $i \in \{1, \dots, p\} \setminus \{j\}$ and $u^j_j = y_j$. After solving each (PS(v, d)), these newly generated upper bounds are added to U and the split upper bound is removed from it. See the Alternative for Line 17 of Algorithm 1.

We apply the following steps for vertex selection using the generated local upper bounds: for each vertex v of the current outer approximation, we first find a subset of the upper bounds among the set U such that $v \leq u$. From this subset, we assign u which has the minimum distance to v as its corresponding upper bound. Note that since $u \in \mathcal{P}$, $\|u - v\|$ yields an upper bound for the distance of v to the upper image. After assigning the corresponding upper bounds, we choose the vertex v^* among the set $V^k \setminus V_{used}$ which has the greatest distance to its assigned upper bound u^*. Clearly, $\|v^* - u^*\|$ yields an upper bound for the approximation error $d_H(P^k, \mathcal{P})$. As it will be detailed in Section 5.5, the algorithms proposed in [7] and [10] also compute bounds for the approximation error during any iteration. Different from them, vertex selection rule (UB) does not solve optimization models to find this approximation error.

Note that some upper bounds have components equal to M. This causes the distances between v and its corresponding upper bound to be significantly large. To tackle this, the modifications are made between lines 10-22 to obtain upper bounds closer to the vertices. While doing that, we benefit from the point $y \in \operatorname{bd} \mathcal{P}$ that is used in order to compute the upper bound. See Procedure 8 for the details.

Alternative for Line 1 of Algorithm 1

```
1: $V_{used} = \emptyset$, $\bar{\mathcal{X}}^0 = \emptyset$, $k = 0$, $U = \{(u, \emptyset)\}$ where $u_i = M$ for $i = 1, \dots, p$.
```

Alternative for Line 17 of Algorithm 1

```
1: Solve (PS($v, d$)). Let $(x^v, z^v)$ be an optimal solution and $y = v + z^v d$.
2: Let $(u, y^u) \in U$ be the corresponding local upper bound returned by SelectVertex
3: for $j \in \{1, \cdots, p\}$ do
4:     $u^j \leftarrow u$, $u^j_j \leftarrow y_j$, $U \leftarrow U \cup \{(u^j, y)\}$
5: end for
6: $U \leftarrow U \setminus \{(u, y^u)\}$
```

Procedure 8 SelectVertex(U,V^k, V_{used})

```
1: $\mathcal{V}_{all} = \emptyset$
2: **for all** $v \in V^k \setminus V_{used}$ **do**
3:    $\mathcal{U}_{temp} = \emptyset$
4:    **for all** $(u, y) \in U$ **do**
5:       **if** $v \leq u$ **then**
6:          Compute $dist_v = \|u - v\|$, $\mathcal{U}_{temp} \leftarrow \mathcal{U}_{temp} \cup \{(v, dist_v, u, y)\}$
7:       **end if**
8:    **end for**
9:    Let $(\hat{v}, \hat{dist}_v, \hat{u}, \hat{y}) \in \arg\min_{(v, dist_v, u, y) \in \mathcal{U}_{temp}} dist_v$
10:   $uTemp \leftarrow \hat{u}$
11:   $l = 0$, $s = 0$
12:   **for** $i \in \{1, \cdots, p\}$ **do**
13:      **if** $\hat{u}_i \neq M$ **then**
14:         $l = l + \hat{y}_i$, $s = s + 1$
15:      **end if**
16:      $val = \frac{l}{s}$
17:   **end for**
18:   **for** $i \in \{1, \cdots, p\}$ **do**
19:      **if** $\hat{u}_i = M$ **then**
20:         $uTemp_i \leftarrow \max\{val, \hat{y}_i\}$
21:      **end if**
22:   **end for**
23:   Compute $\hat{dist}_v = \|uTemp - \hat{v}\|$.
24:   $\mathcal{V}_{all} \leftarrow \mathcal{V}_{all} \cup \{\hat{v}, \hat{dist}_v, \hat{u}\}$
25: **end for**
26: Let $(v^*, dist^*_v, u^*) \in \arg\max_{v \in \mathcal{V}_{all}} \hat{dist}_v$.
27: **return** $v^*$, $u^*$, $dist^*_v$
```

In Section 5.3, we state that we fix (Adj) as the vertex selection rule for the further computational study. When one uses (UB) as the vertex selection rule, we note that it is also possible to use the local upper bounds in order to determine the direction parameter. In particular, for a vertex v with the corresponding local upper bound u, we consider selecting the direction parameter as the vector from v to u. In order to test this direction selection rule we conduct a computational study on Example 3 in which we compare three direction selection rules: $d = \frac{e}{\|e\|}$ (UB-Fixed), $d = \frac{(u-v)}{\|u-v\|}$ (UB-UB), and adjacent vertices approach (UB-Adj). The results are given in Table 5.7. We observe that (UB-Adj) solves less number of scalarizations while (UB-UB) clearly has a negative effect on the runtime and the number of scalarizations to be solved.

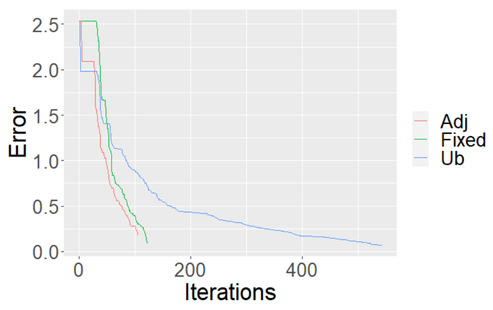
Table 5.7: The effect of different direction selection methods for (UB) on CPU Time and number of scalarization models for Example 3 with $\epsilon = 0.05$

		UB-Fixed	UB-Ub	UB-Adj
$a = 5$	SC	128	547	110
	Time	38.83	173.58	32.51
$a = 7$	SC	127	570	112
	Time	36.81	178.73	33.36
$a = 10$	SC	139	737	90 [1]
	Time	41.20	240.06	26.93
$a = 20$	SC	194	1990	127
	Time	62.16	996.51	37.44

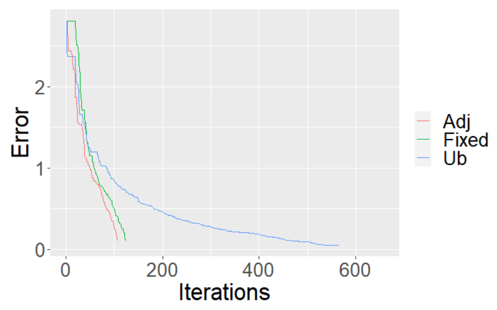
Moreover, in order to observe how fast the bound on the approximation error decreases, we plot the approximation error found by (UB-Adj), (UB-Fixed) and (UB-UB) at each iteration, see Figure 5.5. We confirm that (UB-Adj) terminates faster than (UB-Fixed) and (UB-UB). Based on this preliminary analysis, we decide use (Adj) as the direction selection rule throughout the computational studies in Chapter 6.

[1]For $a = 7$, the outer approximation of (UB-Adj) has less vertices than expected at termination due to some numerical issues in *bensolve tools*.

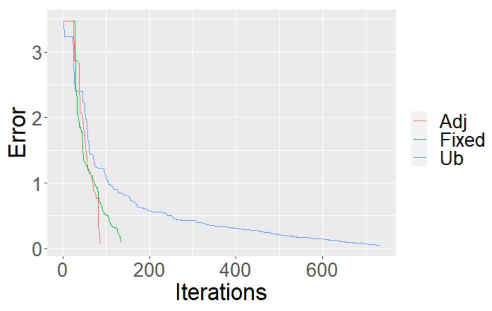
Figure 5.5: The effect of different direction selection methods on the approximation error found by (UB) for Example 3 with $\epsilon = 0.05$



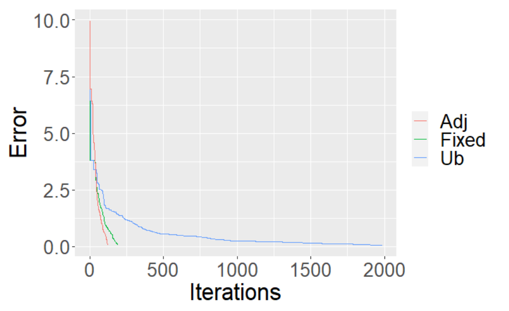
a=5



a=7



a=10



a=20

5.5 Algorithms from the literature

In this section, we briefly explain the similar algorithms from the literature since we will compare our algorithm variants' performances also with them in Chapter 6. The first algorithm, namely Algorithm (KTW) is proposed by Klamroth, Tind and Wiecek in 2003, see [7]; the second one, namely Algorithm (DLSW) is proposed by Dörfler, Löhne, Schneider and Weissing in 2021, see [10]; and the last one, namely Algorithm (AUU), is proposed by Ararat, Ulus and Umer in 2021, see [11].

5.5.1 Algorithm (KTW)

The outer approximation algorithm provided in [7] can be seen as a special case of the general framework given by Algorithm 1, applied to a maximization problem. Here, we shortly explain it for problem (P).

The algorithm assumes that $0 \in \mathcal{P}$. It starts with a polyhedral outer approximation and in iteration k, it minimizes the distance (with respect to an oblique norm which is defined based on a polyhedral Gauge function) between the current outer approximation and the upper image. In order to do that, it solves the following model for every vertex $v \in V^k \setminus V_{used}$:

$$\begin{aligned} \text{maximize } & \lambda \\ \text{subject to } & f(x) \leq \lambda v \\ & x \in \mathcal{X} \\ & \lambda \in \mathbb{R}. \end{aligned} \tag{5.3}$$

Assuming (x^v, λ^v) is an optimal solution to this problem for vertex v, $\min_{v \in V^k \setminus V_{used}} \lambda^v$ yields the oblique-norm-based distance between P^k and $\mathcal{P}$. Then, $v^* \in \arg\min_{v \in V^k \setminus V_{used}} \lambda^v$ is selected in order to update the current outer approximation as in Algorithm 1.

Note that (5.3) is an equivalent problem to (PS(v, d)) with $d = -v$. Indeed, (x^v, λ) is an optimal solution to (5.3) if and only if $(x^v, 1 - \lambda)$ is an optimal solution to (PS($v, -v$)). Note that (KTW) is similar to the algorithm proposed in [8], in the sense that $\hat{p} = 0$ is fixed. However, contrary to the one from [8], the selection of the vertices in each iteration is not arbitrary in (KTW) as explained above. The convergence of (KTW) is shown for two-dimensional problems in [7].

For the computational tests that will be presented in Chapter 6 we take $\hat{p}$ defined by $\hat{p}_i := 2\max\{f_i(x^1), \dots, f_i(x^p)\} - v_i$ for $i = 1, \dots, p$ where v is one of the vertices of P^0 that is fixed at the beginning. Note that this is the same $\hat{p}$ that we used for the fixed point approach (FP) among the direction selection methods, as explained in Section 5.3.3. Then, we modify the model given in (5.3) as:

$$\begin{aligned} &\text{maximize } \lambda && (5.4)\\ &\text{subject to } f(x) \le \hat{p} + \lambda(v - \hat{p}) &&\\ &\qquad\qquad x \in \mathcal{X} &&\\ &\qquad\qquad \lambda \in \mathbb{R}. && \end{aligned}$$

To have a more efficient implementation to select the vertices, we do not solve (5.4) for all $v \in V^k \setminus V_{used}$. Instead, we check if (5.4) is solved in previous iterations. More specifically, if $(v, \cdot, \cdot) \notin V^k_{info}$, we solve (5.4) for v and add (v, y^v, z^v) to V^k_{info} where $y^v := \hat{p} + \lambda^*(v - \hat{p})$ for λ^* being the optimal value of (5.4) and $z^v := \|y^v - v\|$. See Procedure 9 for the details. Note that this procedure makes line 17 of Algorithm 1 redundant, since it is ensured between line 8-10 that $z^v > \epsilon$ when *solve* $= 1$.

Procedure 9 SelectVertex(V^k,V_{used},V^k_{info})

```
 1: $\mathcal{V}^k = \emptyset$
 2: for all $v \in V^k \setminus V_{used}$ do
 3:     if $(v, \cdot, \cdot) \notin V^k_{info}$ then
 4:         Let $(x^v, \lambda^v)$ be a solution to (5.4), $y^v := \hat{p} + \lambda^v(v - \hat{p})$ and $z^v := \|y^v - v\|$.
 5:     else
 6:         Select $(v, y^v, z^v) \in V^k_{info}$
 7:     end if
 8:     $\mathcal{V}^k \leftarrow \mathcal{V}^k \cup \{(v, y^v, z^v)\}$
 9: end for
10: $V^k_{info} \leftarrow \mathcal{V}^k$
11: Let $v^* \in \arg\max_{(v,y^v,z^v)\in\mathcal{V}^k} z^v$
12: return $v^*$, $V^k_{info}$
```

Remark 5.5.1. *Note that for Procedure 9 to work correctly, we modify Algorithm 1 slightly as follows: In line 1, we initialize V^0_{info} as an empty set and in line 19 we also update $V^{k+1}_{info} \leftarrow V^k_{info}$.*

5.5.2 Algorithm (DLSW)

Dörfler et al. [10] recently propose a vertex selection rule that uses the inner approximation obtained in each iteration. Noting that the inner approximation obtained in iteration k is given by $\mathcal{I}^k := \operatorname{cl}\operatorname{conv} f(\mathcal{X}^k) + C$, for each vertex of the current outer approximation P^k, they solve the following model:

$$\begin{aligned} &\text{minimize } \; \|y - v\|^2 \\ &\text{subject to } y \in \mathcal{I}^k \end{aligned} \tag{5.5}$$

For each vertex $v \in V^k \setminus V_{used}$, (DLSW) solves (5.5). Let y^v be an optimal solution of (5.5). A vertex $v^* \in \arg\max_{v \in V^k \setminus V_{used}} \|y^v - v\|$ is selected. Moreover, the direction is fixed as $d := \frac{y^{v^*} - v^*}{\|y^{v^*} - v^*\|}$. Then, an upper bound for the Hausdorff distance between the current outer approximation and the upper image is found as $h = \left\|y^{v^*} - v^*\right\|$, see [10] for the details.

In [10], also an improved version of this algorithm is presented. Accordingly, instead of solving (5.5) for all the vertices V^k of P^k, (DLSW) solves it for a vertex $v^* \in V^k$ only in the following two cases:

- If the problem (5.5) for v^* is not already solved in the previous iterations, that is, if $v^* \notin V^{k-1}$, then the model is solved for v.
- Let x^v satisfy $\mathcal{I}^k = \operatorname{conv} f(\mathcal{X}^k \cup \{x^v\}) + C$, where x^v is optimal for (PS(v, d)) in iteration $k - 1$. If (5.5) is solved for v^* in the previous iterations and $(y^{v^*} - v^*)^\top (f(x^v) - y^{v^*}) \geq 0$, then the solution of (5.5) for v in iteration k and $k - 1$ are the same by [10, Theorem 4.4]. Hence, the quadratic model is solved only if $(y^{v^*} - v^*)^\top (f(x^v) - y^{v^*}) < 0$, see Procedure 10.

Procedure 10 SelectVertex(V^k,V_{used},V^k_{info})

```
1: $\mathcal{V}^k = \emptyset$
2: for all $v \in V^k \setminus V_{used}$ do
3:     if $(v, y^v, z^v) \in V^k_{info}$ then
4:         if $(y^v - v)^\top (f(x^{k-1}) - y^v) < 0$ then
5:             Solve (5.5) and for the vertex $v$ find $y^v \in \mathcal{I}^k$ and let $z^v = \|y - v\|$.
6:         end if
7:     else
8:         Solve (5.5) and for the vertex $v$ find $y^v \in \mathcal{I}^k$ and let $z^v = \|y - v\|$.
9:     end if
10:    $\mathcal{V}^k \leftarrow \mathcal{V}^k \cup \{(v, y^v, z^v)\}$.
11: end for
12: $V^k_{info} \leftarrow \mathcal{V}^k$.
13: Let $v^* \in \arg\max_{(v,y^v,z^v)\in\mathcal{V}^k} z^v$
14: return $v^*$, $V^k_{info}$
```

For Procedure 10 to work correctly, we modify Algorithm 1 as exactly in Remark 5.5.1. Note that line 17 of Algorithm 1 is not provided in [10].

5.5.3 Algorithm (AUU)

Recently, Ararat et al. [11] propose an outer approximation algorithm to solve convex vector optimization problems. Even though the working mechanism of their algorithm is similar to Algorithm 1, it is different since instead of (PS(v, d)) it solves the following norm-minimizing scalarization for a vertex v of P^k:

$$\begin{aligned} \text{minimize } & \|y - v\| \\ \text{subject to } & f(x) \leq_C y \\ & x \in \mathcal{X}, y \in \mathbb{R}^p \end{aligned} \tag{5.6}$$

Note that this scalarization model does not require a direction parameter as (PS(v, d)) does. Instead, it computes the real distance from v to the upper image, which is nothing but $z^v := \|y^v - v\|$, where y^v is an optimal solution. It has been shown in [11] that, the dual of (5.6) also yields a supporting halfspace to the upper image and can be used in order to update the current outer approximation as in line 19 of Algorithm 1. The reader is referred to [11] for the details of the algorithm.

If one solves (5.6) for all the vertices of the current outer approximation P^k, then as it is shown in [11], it possible to compute the exact Hausdorff distance between P^k and $\mathcal{P}$, which is nothing but $\max_{v \in V^k} z^v$, where V^k is the set of vertices of P^k, see also Proposition 3.0.1.

In [11], the vertex selection is arbitrary. Here, for algorithm (AUU), we solve (5.6) for each vertex of P^k and select the farthest vertex to the upper image to proceed with the iteration. This allows the algorithm to provide the approximation error in each iteration rather than at termination. Indeed, this approximation error is equal to the Hausdorff distance between the upper image and the outer approximation as briefly explained above. As in (KTW), the solution of (5.6) for a vertex v is added to V^k_{info}, if (5.6) is not solved for v in the previous iterations. See Procedure 11 for the details.

Remark 5.5.2. *Note that for this algorithm, we use* (5.6) *only for vertex selection. Hence, we still execute Line 14 of Algorithm 1. This means that the algorithm (AUU) is different from the one presented in [11], but it is a variant of Algorithm 1 in which the vertex selection is motivated by [11]. For the Pascoletti-Serafini models, the direction parameter is set such that* $d = \frac{v - y^v}{\|v - y^v\|}$.

Similar to the previous ones, for Procedure 11 to work correctly, we modify Algorithm 1 as exactly in Remark 5.5.1.

Procedure 11 SelectVertex(V^k,V_{used},V^k_{info})

```
1: $\mathcal{V}^k = \emptyset$
2: for all $v \in V^k \setminus V_{used}$ do
3:     if $(v, \cdot, \cdot) \notin V^k_{info}$ then
4:         Let $y^v$ be a solution to (5.6) and $z^v := \|y^v - v\|$
5:     else
6:         Select $(v, y^v, z^v) \in V^k_{info}$
7:     end if
8:     $\mathcal{V}^k \leftarrow \mathcal{V}^k \cup \{(v, y^v, z^v)\}$
9: end for
10: $\mathcal{V}^k_{info} \leftarrow \mathcal{V}^k$
11: Let $v^* \in \arg\max_{(v,y^v,z^v)\in\mathcal{V}^k} z^v$
12: return $v^*$, $V^k_{info}$
```

Chapter 6

Computational Results

In this chapter, we conduct a computational study on the examples listed in Section 5.2. We compare the proposed variants with the algorithms discussed in Section 5.5. We use three stopping criteria: the approximation error, CPU time and cardinality of the solution set and the results are given in Sections 6.2, 6.3 and 6.4, respectively.

The algorithms are implemented using MATLAB R2020b. The scalarizations are solved via CVX v2.2 [21], [22] and SeDuMi 1.3.4 [23]. Moreover, *bensolve tools* [16] is used to solve vertex enumeration problems. The computer specification that is used throughout the computational study is Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz.

Recall that depending on our preliminary analysis from Section 5.3.4, we fix the direction selection rule as (Adj), see Section 5.3.1. Together with this direction selection rule, we consider all vertex selection rules, namely (C), (Adj) and (UB). In addition, we also consider the random vertex selection (R) because of its good performance in our preliminary analysis in Section 5.3.4. Since the direction selection rule (Adj) is common for all the variants that we will compare, we will not indicate it and simply call the proposed variants: (C), (Adj), (UB) and (R). For the variants that fix (R) as a vertex selection rule, we solve each problem five times, and we report the average of the results obtained from these five runs. Before presenting our computational study, we discuss the two proximity measures that we use in this chapter to compare the variants.

6.1 Proximity Measures

The proximity performance measures evaluate how well the approximation of a solution set represents the whole set of solutions. Below we describe two measures to compare the proximity of the algorithms: the approximation error and the hypervolume gap. The former measure uses the upper image $\mathcal{P}$ together with its outer approximation, while the latter measure benefits from the gap between inner and outer approximations.

6.1.1 Approximation Error

For the outer approximation algorithms described in this thesis, the approximation error can be defined as the Hausdorff distance between the upper image and the outer approximation P_K of the upper image that is returned by the algorithm, namely, $d_H(P^K, \mathcal{P})$. where P^K is the outer approximation .

Recall that (AUU) calculates the exact $d_H(P^K, \mathcal{P})$ by its structure while (KTW), (DLSW), and (UB) provide upper bounds for this approximation error. In order to compute the correct Hausdorff distance if it is not available, we solve the following model for all the vertices $v \in V^K$, where V^K is the set of vertices of P^K.

$$\begin{aligned} \text{minimize } & \|p - v\| \\ \text{subject to } & p \in \mathcal{P} \end{aligned} \tag{6.1}$$

The optimal objective value d^v of this problem is the distance from vertex v to $\mathcal{P}$. Since $P^K \supseteq \mathcal{P}$ and $\operatorname{recc} P^K = \operatorname{recc} \mathcal{P}$, we have $\max_{v \in V^K} d^v = d_H(P^K, \mathcal{P})$ by Proposition 3.0.1.

6.1.2 Hypervolume Gap

Let $u \in \mathcal{P}$ be a sufficiently large upper bound for the image of the feasible region in the sense that $\{u\} - C \supseteq f(\mathcal{X})$ and V^K be the vertices of the outer approximation at the termination of the algorithm. Let V_1 be the hypervolume of the polytope defined as $(\operatorname{conv} f(\bar{\mathcal{X}}^K) + C) \cap (\{u\} - C)$ and let V_2 be the hypervolume of the polytope $(\operatorname{conv} V^K + C) \cap (\{u\} - C)$. Then the hypervolume gap is calculated as $V_h = V_2 - V_1$.[1] Note that for all the test examples given in Section 5.2, the ordering cone is the positive orthant, $C = \mathbb{R}^p_+$. Hence, in order to find the problem specific upper bound u for these examples, we solve the following model for each objective $f_i(x)$ where $i = 1, \ldots, p$.

$$\begin{aligned} &\text{maximize } f_i(x) \\ &\text{subject to } x \in \mathcal{X} \end{aligned} \tag{6.2}$$

Then, the upper bound is set as u where u_i is the optimal objective value of this model. For the examples where (6.2) cannot be solved (in particular for Example 2), the upper bound is selected as $u_i = \max f_i(\bar{\mathcal{X}}^{K_1} \cup \ldots \cup \bar{\mathcal{X}}^{K_s})$, where s is the number of the variants of Algorithm 1 that we solve the problem and $\bar{\mathcal{X}}^{K_i}$ is the solution set returned by the i^{th} variant for all $i \in \{1, \ldots, s\}$. Note that, the solutions found by the weighted-sum scalarizations at initialization are eliminated from $\bar{\mathcal{X}}^{K_i}$ when computing u_i. This is because these solutions are excessively large in one component and this causes *bensolve tools* not to perform vertex enumeration.

We use *bensolve tools* to find the vertices of the polytopes and convhulln() function in MATLAB to find the hypervolume of the polytopes. The hypervolume gaps that could not be calculated by *bensolve tools* are given as (-) in the tables.

[1] I would like to thank Çağın Ararat and Simay Tekgül for sharing their insightful ideas on Hypervolume Gap calculations.

6.2 Computational Results Based on Approximation Error

For the computations in this section, we solve the examples with a predetermined approximation error $\epsilon > 0$. Accordingly, when the algorithms terminate it is guaranteed that the Hausdorff distance between the outer approximation and the upper image is less than ϵ, that is $d_H(P^K, \mathcal{P}) < \epsilon$. For each example, we report the total number of models to choose a vertex (VS), the number of scalarizations (SC) solved throughout the algorithm, the CPU time (Time), the realized approximation error (Error), and the hypervolume gap (HG). Note that VS is positive only for the algorithms in Section 5.5 as the others do not solve models to select vertices. The realized approximation is error is the error that is returned by the algorithm and it is clearly less than ϵ. Tables 6.1-6.5 show the results for Examples 1-5, respectively.

Note that for Example 4, we do not report the results for Algorithms (AUU), (DLSW) and (KTW), see Table 6.4. This is because the *bensolve tools* was unable to perform the vertex enumeration in some iteration of these algorithms. Moreover, In Table 6.5, (DLSW) results are not reported. This is because solver failures occurred for these instances. Since the upper images are polyhedral for the linear problems in Example 4, the algorithms have the potential to return the exact upper image. For our numerical tests, we take ϵ sufficiently small (10^{-8}) for these problems. Accordingly, we did not report the realized error and the hypervolume gap in Table 6.5 because the proximity measures are negligibly small. Finally, (KTW) could not solve Example 2. Hence, Table 6.2 does not show the corresponding results. This will be explained as a separate case later.

Table 6.1: Computational results for Example 1 ($\epsilon = 0.005$ and $\epsilon = 0.05$ for p=3 and p=4, respectively)

p	Measures	AUU	DLSW	KTW	UB	R	C	Adj
	VS	572	1281	901	0	0	0	0
	SC	124	239	143	389	382.2	414	406
3	Time	206.63	355.21	256.86	113.15	110.33	120.94	122.51
	Error	0.0049	0.005	0.005	0.005	0.0049	0.0046	0.0038
	HG	0.0247	0.0095	0.0141	0.0111	0.0104	0.0118	0.0117
	VS	515	2192	1735	0	0	0	0
	SC	62	128	74	496	449.8	485	462
4	Time	185.94	517.33	524.39	171.27	145.61	154.83	147.07
	Error	0.0497	0.0499	0.048	0.05	0.046	0.0491	0.0439
	HG	0.4366	-	0.4146	0.244	0.2593	0.2593	0.2504

Table 6.2: Computational results for Example 2 ($\epsilon = 0.005$).

Measure	AUU	DLSW	UB	R	C	Adj
VS	387	414	0	0	0	0
SC	39	57	103	120.8	112	351
Time	141.46	145.25	33.18	37.65	34.75	107.51
Error	0.0049	0.0049	0.0046	0.0043	0.0043	0.0046
HG	2.1795	0.5875	0.9108	0.9166	0.7918	0.9107

Table 6.3: Computational results for Example 3 ($\epsilon = 0.05$).

a	Measure	AUU	DLSW [2]	KTW	UB	R	C	Adj
5	VS	160	1531	299	0	0	0	0
	SC	43	66	63	110	102.2	112	101
	Time	62.21	252.83	84.64	32.51	29.49	31.8	40.26
	Error	0.0472	0.0499	0.0489	0.0355	0.0465	0.0337	0.05
	HG	2.0076	1.5598	0.8997	1.1933	1.3571	1.2773	1.4888
7	VS	169	330	373	0	0	0	0
	SC	46	75	70	112	112.6	113	119
	Time	62.7	86.36	104.72	33.36	32.41	32.57	37.11
	Error	0.0472	0.0468	0.0431	0.0485	0.0428	0.0456	0.0484
	HG	2.8207	1.4369	1.3481	2.0808	3.394	2.9109	1.824
10	VS	170	1060	327	0	0	0	0
	SC	47	72	76* [3]	90*	120.8	126	115*
	Time	64.01	168.61	93.49	26.93	34.88	36.18	40.66
	Error	0.0498	0.0497	0.0431	0.0304	0.0418	0.0422	0.0482
	HG	4.1825	2.299	-	-	5.0066	3.9589	-
20	VS	198	323	1764	0	0	0	0
	SC	49	77	157	127	133.2	148	102*
	Time	73.11	85.92	499.27	37.44	38.72	42.88	40.05
	Error	0.0486	0.049	0.0499	0.047	0.0425	0.0421	0.0455
	HG	8.7898	5.0659	1.5611	7.0989	4.8624	4.8074	-

[2] For (DLSW), the instance $a = 5$ and $a = 10$ of Example 3 could not be solved due to the limitations of *bensolve tools*. Hence, we benefit from the problem symmetry, that is the problem is scaled along different axes for these instances. I would like to thank Daniel Dörfler for the useful recommendation.

[3] The outer approximation of the instances which are indicated by * in the table terminated with less vertices than expected due to some numerical issues in *bensolve tools*.

Table 6.4: Computational results for Example 4 ($\epsilon = 0.05$)

a	Measure	UB	R	C	Adj
5	SC	1246	1085	1163	1108
	Time	827.73	515.53	598.77	516.48
	Error	0.0489	0.0490	0.0494	0.0490
	HG	3.9178	3.9733	4.2713	3.9686
7	SC	1295	1137.8	1203	1203
	Time	909.46	550.34	579.92	580.53
	Error	0.0492	0.0489	0.0469	0.0479
	HG	-	-	-	-
10	SC	1386	1286	1183	1290
	Time	1019.63	592.55	529.31	603.17
	Error	0.0462	0.0497	0.04895	0.0455
	HG	-	-	-	-

Table 6.5: Computational results for Example 5 ($\epsilon = 10^{-8}$)

p	Inst.	Measure	AUU	KTW	UB	R	C	Adj
3	Avg.	VS	602.45	1139.5	0	0	0	0
		SC	99.6	96.85	226.65	195.2	231.55	307.45
		Time	196.72	291.07	61.33	51.32	63.89	80.46
4	1	VS	5610	14914	0	0	0	0
		SC	275	263	861	517.6	1180	1103
		Time	1671.88	3762.82	319.72	177.42	385.21	338.85
	2	VS	667	1174	0	0	0	0
		SC	80	71	161	125.8	124	147
		Time	233.4	291.53	47.08	32.92	33.29	41.75
	3	VS	2557	6531	0	0	0	0
		SC	173	157	427	331	427	635
		Time	783.09	1645.65	137.02	92.86	129.12	181.58

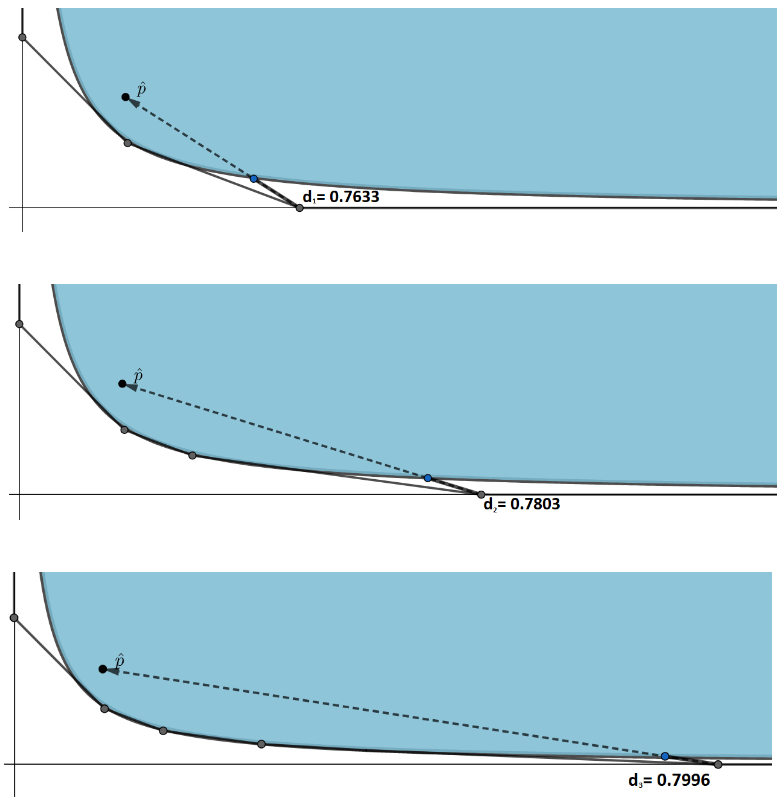
When Tables 6.1, 6.2, 6.3, 6.5 are analyzed, (AUU) seems to have shorter CPU time compared to (DLSW) and (KTW). Moreover, (UB), (R), (C), and (Adj) are faster than (AUU), (DLSW), and (KTW) in all examples. When we compare the proposed variants, we observe that (R) generally provides better performance in terms of runtime for the linear problems, see Table 6.5. Moreover, for nonlinear examples with $p = 4$, namely Example 4, (UB) is outperformed by the rest of the variants. For the rest of the examples, (UB), (R), (C), and (Adj) have similar performances in terms of runtime. There is an exception in Example 2, where (Adj) is outperformed by (UB), (R), and (C) with a significant difference. Yet, (Adj) still has a shorter CPU time than (AUU), (DLSW), and (KTW).

Since the stopping criteria is the approximation error, the realized errors are close to each other for all the algorithms as expected. When the two proximity measures are compared, we observe that approximation error and hypervolume gap are not necessarily correlated in each example, see for instance Table 6.3 where $a = 20$. Here, the approximation errors are similar for all the variants, but the hypervolume gaps differ significantly.

Now, we will explain why (KTW) can not solve Example 2. Recall that for this example the feasible region is not compact. Indeed, the upper image has an asymptotic behavior in a sense that the boundary of $\mathcal{P}$ approaches to the axes as it gets further away from the origin. Because of this structure of the upper image, (KTW) tends to find larger distances between a vertex and the corresponding point on the boundary of $\mathcal{P}$. In order to illustrate this behavior, we pick $\hat{p}$ as $(1.2, 1.2)^\top$ and perform three iterations of (KTW). As it can be seen from Figure 6.1, even though the considered vertex, say v, gets closer to the upper image, $\|y^v - v\|$ increase. Note that for illustrative purposes, $\hat{p}$ is selected as a point that is already close to the boundary of $\mathcal{P}$ here. However, this behavior of the algorithm may be observed for any $\hat{p}$, but generally after a certain number of iterations.

In our computations, we did not observe this behavior illustrated in Figure 6.1 mainly because we fix $\hat{p}$ large enough. However, since $\hat{p}$ is large, it caused numerical issues.

Figure 6.1: KTW iterations for Example 2



6.3 Computational Results Under Limited Runtime

In this section, we run the same examples under time limit and we compare the proximity measures that the algorithms yield. In particular, for each example we give a runtime limit to the variants so that they terminate before reaching the approximation error that we set for our previous set of experiments from Section 6.2. For each Example, the time limits are determined by considering the variant with the shortest CPU time. Around half of this CPU time is set as the time limit.

In Example 5, for $p = 3$, we solve the same randomly generated instances that we used in Section 6.2. The CPU times for these instances differ significantly. Hence, the most time consuming 8 instances are selected, and the same runtime limit of 50 seconds is fixed for these instances. The results in Table 6.10 are the averages over the 8 instances.

For the variants (UB), (R), (C), and (Adj), two different cardinalities are given in the tables. The greater cardinality is calculated by $|\mathcal{X}^K|$, where $\mathcal{X}^K$ is the solution set when the algorithm is terminated. The smaller cardinality is found by using Remark 5.1.1, that is, by adding the solution x^v to the solution set only if the corresponding z^v satisfies $z^v \leq \epsilon$. Note that the greater cardinality for variants (UB), (R) (C) and (Adj) and the cardinality for (AUU), (DLSW) and (KTW) is exactly the same as the number of scalarizations, which is denoted by (SC) in Section 6.2.

When terminated after a certain time limit, some of the variants are not able to give the approximation error corresponding to the returned solution set. Hence, for the tables in this section, we calculate the approximation error as discussed in Section 6.1.1.

Table 6.6: Computational results under limited runtime for Example 1 (50 seconds for $p = 3$, 75 seconds for $p = 4$)

p	Measure	AUU	DLSW	KTW	UB	R	C	Adj
3	VS	95	139	129	0	0	0	0
	Error	0.0168	0.0136	0.0183	0.0057	0.0067	0.0054	0.0071
	HG	0.1335	0.0921	0.1239	0.0442	0.1417	0.0898	0.1289
	Cardinality	29	36	27	124 / 35	127.8 / 44.4	120 / 32	124 / 32
4	VS	154	234	183	0	0	0	0
	Error	0.0783	0.0707	0.0771	0.0348	0.0338	0.0356	0.0359
	HG	-	1.0732	-	0.5305	0.5523	0.5557	0.6627
	Cardinality	28	32	30	155 / 80	169.2 / 98.8	161 / 95	166 / 94

Table 6.7: Computational results under limited runtime for Example 2 (15 seconds)

Measure	AUU	DLSW	UB	R	C	Adj
VS	31	37	0	0	0	0
Error	0.0296	0.0346	0.0283	0.0589	0.0129	0.0989
HG	32.369	34.5366	0.8966	39.8425	12.4551	1.7888
Cardinality	15	15	44 / 16	45 / 16.2	42 / 11	46 / 27

Table 6.8: Computational results under limited runtime for Example 3 (15 seconds)

a	Measure	AUU	DLSW	KTW	UB	R	C	Adj
5	VS	25	42	35	0	0	0	0
	Error	0.4119	0.4625	0.2274	0.1744	0.5984	**0.1057**	0.2059
	HG	18.37192	23.3427	8.4821	2.7629	5.2119	2.5381	3.0768
	Cardinality	10	9	12	34 / 14	35.8 / 10	35 / 8	31 / 10
7	VS	28	33	34	0	0	0	0
	Error	0.4138	0.6168	0.1839	0.2163	0.2862	**0.1494**	0.4557
	HG	25.64368	19.7291	5.8315	4.0851	8.3792	4.9919	6.1088
	Cardinality	11	12	16	35 / 9	36.4 / 9.2	36 / 10	35 / 14
10	VS	28	42	37	0	0	0	0
	Error	0.4033	0.4424	0.1953	0.48	0.8142	0.1672	0.185
	HG	38.8893	-	10.3223	-	18.7573	5.38	5.9085
	Cardinality	11	12	16	36 / 9	37 / 11.2	37 / 7	34 / 11
20	VS	25	35	36	0	0	0	0
	Error	0.6616	0.2958	0.3164	0.3712	0.4522	**0.1878**	0.5074
	HG	100.5123	59.1384	25.3849	11.9967	33.3651	12.4483	11.5402
	Cardinality	11	13	16	37 / 6	37.4 / 11.8	37 / 10	37 / 9

Table 6.9: Computational results under limited runtime for Example 4 (250 seconds)

a	Measure	UB	R	C	Adj
5	VS	0	0	0	0
	Error	0.101	0.2072	0.1678	0.2072
	HG	-	-	-	-
	Cardinality	392 / 233	508.5 / 164.5	452 / 304	537 / 378
7	VS	0	0	0	0
	Error	0.1261	0.1825	0.1013	0.1312
	HG	7.2337	-	-	-
	Cardinality	366 / 211	480.4 / 318.4	424 / 276	504 / 349
10	VS	0	0	0	0
	Error	0.101	0.2759	0.3343	0.1518
	HG	-	-	-	-
	Cardinality	382 / 214	498.6 / 329	519 / 361	513 / 351

Table 6.10: Computational results under limited runtime for Example 5 (50 seconds for $p = 3$, 100 seconds in instance 1, 15 seconds in instance 2, 50 seconds in instance 3 for $p = 4$)

p	Inst.	Measure	AUU	DLSW	KTW	UB	R	C	Adj
3	Avg.	VS	69.125	72.375	98	0	0	0	0
		Error	1.3145	2.1944	0.9805	1.1396	2.8557	0.9553	1.0659
		HG	15966.61	22167.71	11919.47	32596.53	3564.33	2825.69	3209.04
		Cardinality	22.13	20.25	26.25	95.38 / 25.63	91.75 / 22.13	89 / 21	92.5 / 29.63
4	1	VS	318	338	403	0	0	0	0
		Error	2.7899	3.1038	4.8830	1.2209	2.5621	1.4384	1.8833
		HG	3453462.73	2776469.82	3176108.34	-	-	139495.18	268168.59
		Cardinality	38	33	30	269 / 101	286.4 / 82.6	266 / 111	336 / 182
	2	VS	43	41	57	0	0	0	0
		Error	11.3862	5.644	10.2325	3.048	8.6336	1.7226	**0.8027**
		HG	415587.08	224514.46	182864.48	2731485.63	23599.91	13700.71	19003.47
		Cardinality	12	13	13	56 / 20	55.4 / 14.4	55 / 18	57 / 20
	3	VS	182	175	194	0	0	0	0
		Error	2.9557	3.2645	1.5588	1.0328	3.1134	**0.8621**	1.1001
		HG	722544.84	803350.19	408030.61	37740.57	42125.39	14692067.39	46920.04
		Cardinality	26	21	30	165 / 66	163.8 / 44	150 / 50	180 / 88

In Examples 2 and 3, (C) returned smaller approximation error compared to the other variants in general, see Tables 6.7 and 6.8. The worst approximation errors in each row is indicated by grey color if it is significantly higher than the rest. Observe that in the tables, this is obtained by (AUU), (DLSW), (KTW), and (R) for different problem settings.

We observe that the cardinalities returned by the variants (when Remark 5.1.1 is applied for the proposed variants) are similar over all variants for two and three-dimensional examples. For four-dimensional examples, the cardinalities returned by (AUU), (DLSW), and (KTW) are smaller than the rest of the variants. Moreover, the approximation errors of (AUU), (DLSW), and (KTW) are larger than (UB), (C), and (Adj) for these cases. (R) behaves similar to (AUU), (DLSW), and (KTW) for these runs. See Tables 6.6 and 6.10.

An additional computational study is conducted on Example 3 to observe the behavior of the algorithms as the time limit changes. The variants are terminated after 50 seconds, 100 seconds and 200 seconds. In this set of experiments, we also observe the effect of ϵ used within the variants (UB), (C), (Adj) and (R) even if the approximation error is not the stopping condition. Recall that in these variants, we do not necessarily select the vertex which is the farthest away from the upper image. Hence ϵ has still an important role in order to decide if the algorithm would add a cut to the current outer approximation or not. In order to observe this effect we run all the variants for the above mentioned time limits both with $\epsilon = 0.005$ and $\epsilon = 0$.

The results are given in the Appendix, see Tables A.1-A.3. As expected, the performance of (AUU), (KTW), and (DLSW) are not affected by the change in ϵ, since they choose the farthest point based on the models that they solve for vertex selection. On the other hand, an increase in ϵ generally affects the performance of (UB), (C), and (Adj) positively. They reach smaller approximation errors when $\epsilon = 0.005$, compared to $\epsilon = 0$ after 200 seconds. (UB) is affected significantly by the change in ϵ compared to (R), (C), and (Adj). Indeed, the approximation error obtained by (UB) is an upper bound to that obtained by all variants for $\epsilon = 0$. This can be explained with (UB)'s tendency to choose the vertices that are closer to the tails of the boundary of the upper image. See Figures 6.2 and 6.3 for the approximation errors obtained from all the variants for $a = 5$.

We also plot the hypervolume gaps that are obtained by each variant for these experiments, see Figures 6.4 and 6.5. We observe that (C), and (Adj) have smaller hypervolume gaps than (AUU), (KTW), and (DLSW). This may be explained by the fact that the solution sets of (C), and (Adj) have greater cardinality compared to the other variants, see Tables A.1-A.3. However, (UB) and (R) do not provide low hypervolume gaps despite having a large cardinality when $\epsilon = 0$.

Figure 6.2: Approximation Error vs Time ϵ=0, Example 3, a=5

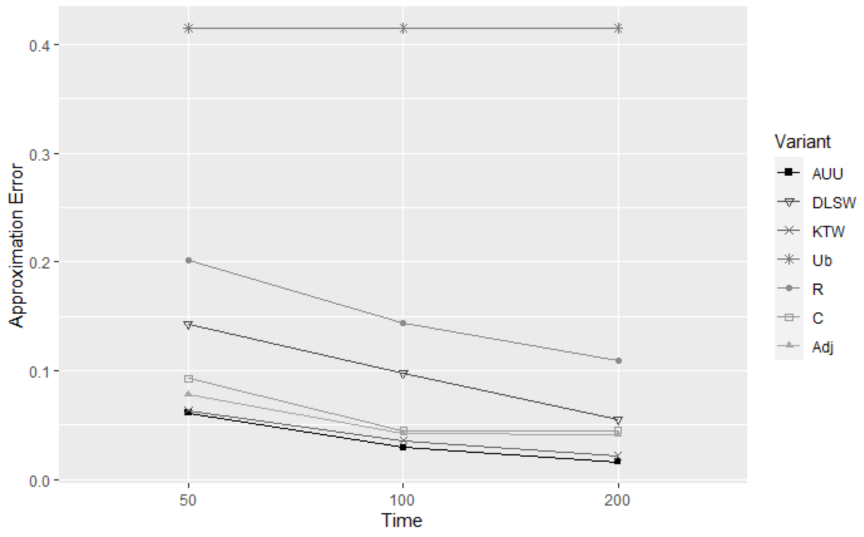


Figure 6.3: Approximation Error vs Time ϵ=0.005, Example 3, a=5

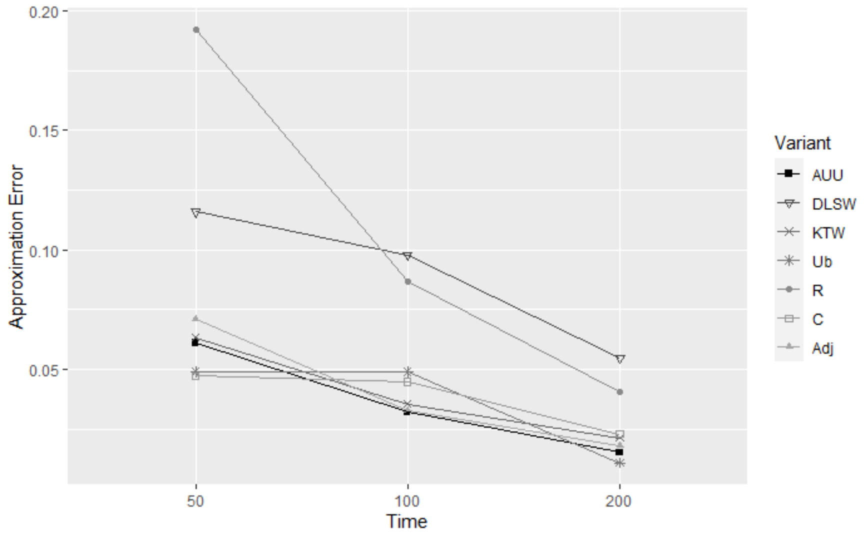


Figure 6.4: Hypervolume Gap vs Time ϵ=0, Example 3, a=5

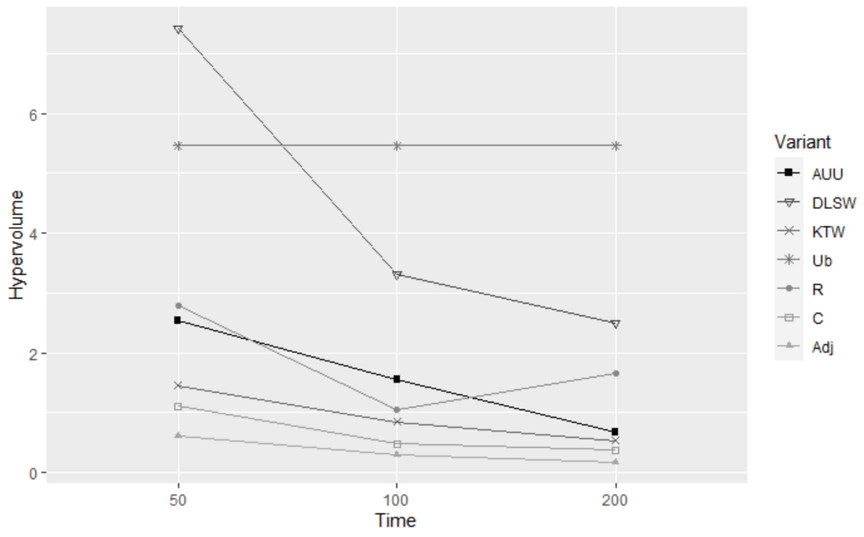
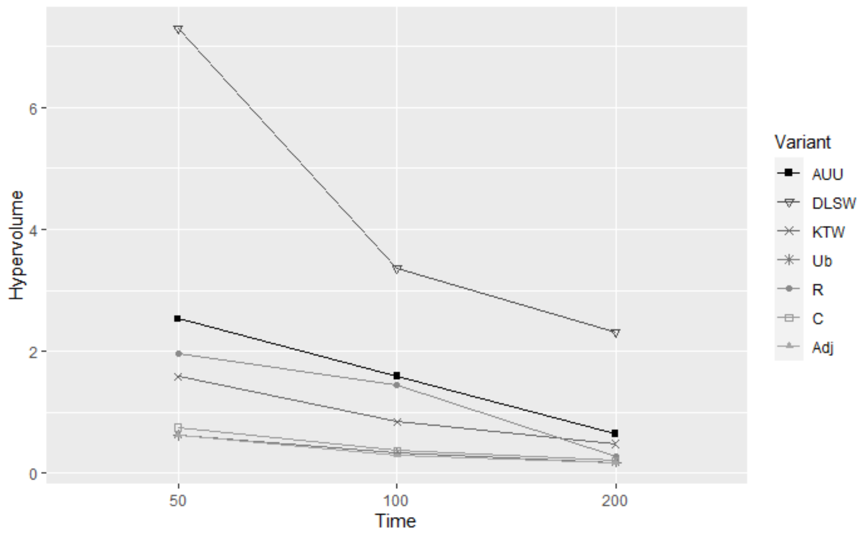


Figure 6.5: Hypervolume Gap vs Time ϵ=0.005, Example 3, a=5



6.4 Computational Results for Fixed Cardinality

In this section, we compare the performances of the algorithms when they find a fixed number of solutions, that is, when they run until they find a solution set with a predetermined cardinality. Similar to the idea in Section 6.3, for each example we give a cardinality limit to the variants so that they terminate before reaching the approximation error that is set for the experiments in Section 6.2. The cardinalities are selected considering the number of scalarizations (SC) that are obtained in Section 6.2. In Example 5 where $p = 3$, the same 8 instances are selected as in Section 6.3, and the cardinality of the solution set ($|\bar{\mathcal{X}}|$) are fixed as 50 over 8 for all these instances. For the variants (UB), (R), (C) and (Adj), cardinalities are computed based on Remark 5.1.1. Similar to the case in Section 6.2, for the variants which do not return the approximation error, we calculate it as explained in Section 6.1.1. Tables 6.11-6.15 to show the results for Examples 1-5, respectively.

Table 6.11: Computational results with fixed cardinality for Example 1

$\lvert\bar{\mathcal{X}}\rvert$	p	Measure	AUU	DLSW	KTW	UB	R	C	Adj
100	3	VS	451	455	695	0	0	0	0
		Time	252.12	175.98	266.51	93.74	51.98	90.66	84.96
		Error	0.0062	0.0065	0.0067	0.0196	0.0325	0.0195	0.0116
		HG	0.02836	0.0293	0.0276	0.0337	0.0673	0.0685	0.0367
50	4	VS	401	430	816	0	0	0	0
		Time	197.53	144.85	327.38	62.2	48.24	54.12	48.75
		Error	0.0578	0.0579	0.0614	0.1448	0.4142	0.0894	0.1281
		HG	0.5136	0.5191	-	-	1.1331	0.4865	1.0227

Table 6.12: Computational results with fixed cardinality for Example 2

$\|\mathcal{X}\|$	Measure	AUU	DLSW	UB	R	C	Adj
20	VS	64	90	0	0	0	0
	Time	29.15	34.97	15.83	16.81	21.45	11.72
	Error	0.0197	0.0188	0.0283	0.2236	0.0129	0.0989
	HG	14.9805	16.0643	11.7677	32.6323	81.5325	-

Table 6.13: Computational results with fixed cardinality for Example 3

$\|\mathcal{X}\|$	a	Measure	AUU	DLSW	KTW	UB	R	C	Adj
30	5	VS	98	581	106	0	0	0	0
		Time	50.51	174.8	106.15	25.14	25.27	24.32	23.34
		Error	0.0789	0.0979	0.0543	0.0629	0.1752	0.1006	0.102
		HG	3.8827	4.2174	1.9568	2.9361	6.8825	3.6277	4.3917
	7	VS	100	106	102	0	0	0	0
		Time	50.48	41.22	78.82	25.78	24.84	25.87	21.31
		Error	0.0789	0.1112	0.0682	0.2163	0.1677	0.1184	0.4557
		HG	4.3195	4.8558	2.8265	6.3619	8.9743	5.7084	7.8197
	10	VS	104	505	97	0	0	0	0
		Time	52.06	150.75	71.16	26.66	25.59	26.1	24.69
		Error	0.0874	0.1122	0.0639	0.1056	0.2086	0.1105	0.1535
		HG	8.3705	7.7040	4.0736	6.4323	13.3795	6.784	11.5452
	20	VS	123	104	116	0	0	0	0
		Time	59.48	41.4	76.91	26.85	26.26	25.18	24.97
		Error	0.0877	0.1264	0.0945	0.1256	0.3129	0.1827	0.5069
		HG	15.7741	15.6121	10.2093	7.2898	15.9299	7.97562	7.8440

Table 6.14: Computational results with fixed cardinality for Example 4

$\lvert\mathcal{X}\rvert$	a	Measure	UB	R	C	Adj
100	5	Time	137.56	112.89	142.94	100.93
		Error	0.1221	0.4142	0.1039	0.1750
		HG	-	16.2228	9.2798	7.4764
	7	Time	131.28	113.22	148.59	99.75
		Error	0.1261	0.3092	0.1530	0.2072
		HG	-	-	17.5413	-
	10	Time	145.59	115.72	129.40	113.33
		Error	0.2072	0.4101	0.3343	0.2226
		HG	-	-	-	-

Table 6.15: Computational results with fixed cardinality for Example 5

p	$\lvert\mathcal{X}\rvert$	Inst.	Measure	AUU	DLSW	KTW	UB	R	C	Adj
3	50	Avg.	VS	204	233	347.75	0	0	0	0
			Time	72.5315	81.9848	94.2608	41.8377	43.3095	46.1665	36.0808
			Error	0.2502	0.4715	0.2564	0.3672	1.0093	0.4707	0.6722
			HG	5573.212	3567.388	4483.984	1025.182	1325.694	1051.39	1860.854
4	125	1	VS	2533	3042	6457	0	0	0	0
			Time	768.6969	904.62	1641.782	110.6258	127.411	106.6825	79.8264
			Error	0.229	0.3482	0.2559	1.2209	1.2063	0.4497	1.8833
			HG	606958.2	163459.1	605558.1	163636.9	40282.13	118499.2	331493
	40	2	VS	266	298	609	0	0	0	0
			Time	87.0934	95.8139	156.7077	25.1622	26.8509	26.8377	24.3552
			Error	0.4871	0.3892	0.2523	0.4871	0.3716	0.2489	0.1966
			HG	30354.16	12517.19	24715.21	4128.84	1112.319	876.6259	2709358
	75	3	VS	1207	2027	1167	0	0	0	0
			Time	386.5177	363.6088	534.5691	58.706	65.5746	63.5133	47.5789
			Error	0.4436	0.5809	0.3119	1.0328	0.9916	0.8274	1.1001
			HG	121479.4	79236.24	138802.3	27890.16	10856.6	89151.33	54859.54

From Tables 6.11-6.15, we observe that the variants (UB), (R), (C) and (Adj) require less CPU time compared to (AUU), (DLSW) and (KTW) also in order to find a solution set with fixed cardinality. Note that this is in line with the observations from Section 6.2. Among the algorithms from Section 5.5, (DLSW) requires less CPU time compared to (AUU) and (KTW), especially if the dimension of the objective space is high, see Tables 6.11 and 6.15, instance 3 where p=4. On the other hand, when we compare the proposed variants, we see that (Adj) is slightly faster than the others in most examples, while the difference is more significant in some examples, see for instance Tables 6.14 and 6.15.

Moreover, we see that in general, approximation errors found by (AUU), (DLSW) and (KTW) are smaller than or very close to those found by (UB), (R), (C) and (Adj). This shows the trade-off between the runtime and the approximation error. When we compare the algorithms from the literature, we see that under cardinality limit, (AUU) and (KTW) yield better results in terms of the approximation error compared to (DLSW), in general. On the other hand, the variants (UB), (R), (C) and (Adj) are comparable in that sense.

Hypervolume gaps are comparable among the variants especially when $p = 3$. For instance, In Example 1, (AUU), (DLSW) and (KTW) have slightly smaller hypervolume gaps compared to the proposed variants, while (UB), (R), (C) and (Adj) have smaller hypervolume gaps in Example 5, both with $p = 3$. When we check the results for $p = 4$, in general, we observe a similar behavior as the approximation errors.

Remark 6.4.1. *During the implementation of the codes, two vertices v^1 and v^2 are treated to be the same if $\|v^1 - v^2\| \leq 10^{-10}$. These precision levels may have an impact on the performances of the variants.*

Chapter 7

Conclusion and Future Work

We present a general framework of outer approximation algorithms to solve CVOPs where Pascoletti-Serafini scalarization is used. We propose different methods to select the two parameters of the scalarization problem. First, we compare different direction selection rules. We observe that the direction selection rule (Adj) generally shows promising results as it increases the performance of the algorithm by decreasing the runtime and the number of scalarizations to be solved when different simple vertex selection rules are applied. We also propose additional vertex selection rules that can be used along with (Adj).

In addition to the proposed variants of this algorithm, we implement three relevant algorithms from the literature. We provide an extensive computational study to compare the variants / algorithms. We observe that the vertex selection methods that do not require solving additional models perform better in terms of CPU time when the stopping condition is the approximation error and cardinality. Under limited runtime, the proposed variants' proximity measures are better or comparable with the algorithms from the literature. We also observe that the selection of ϵ affects the performance of the proposed variants, while the algorithms in Section 5.5 are not affected by the changes in ϵ by their structure. Moreover, the sizes of the solution sets for the algorithms in Section 5.5 are generally smaller, so they find coarser set of solutions.

As a future work, different measures can be utilized to compare the quality of the solution sets of the variants. Distribution and spread measures such as uniformity, evenness or distribution metric can be useful for this comparison. See Appendix A.3 for a discussion on these measures.

Bibliography

[1] H. Markowitz, "Portfolio selection," *The Journal of Finance*, vol. 7, no. 1, pp. 77–91, 1952.

[2] B. Rudloff and F. Ulus, "Certainty equivalent and utility indifference pricing for incomplete preferences via convex vector optimization," *Mathematics and Financial Economics*, vol. 15, no. 2, pp. 397–430, 2021.

[3] A. Hamel, B. Rudloff, and M. Yankova, "Risk minimization and set-valued average value at risk via linear vector optimization," 01 2012.

[4] Ç. Ararat, O. Çavuş, and A. I. Mahmutoğulları, "Multi-objective risk-averse two-stage stochastic programming problems," 2017.

[5] S. Gass and T. Saaty, "The computational algorithm for the parametric objective function," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 39–45, 1955.

[6] A. Pascoletti and P. Serafini, "Scalarizing vector optimization problems," *Journal of Optimization Theory and Applications*, vol. 42, no. 4, pp. 499–524, 1984.

[7] T. J. Klamroth, K. and M. M. Wiecek, "Unbiased approximation in multicriteria optimization," *Mathematical Methods of Operations Research*, vol. 56, pp. 413–437, 2003.

[8] 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.

[9] 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.

[10] D. Dörfler, A. Löhne, C. Schneider, and B. Weißing, "A benson-type algorithm for bounded convex vector optimization problems with vertex selection," *Optimization Methods and Software*, 2021.

[11] Ç. Ararat, F. Ulus, and M. Umer, "A norm minimization based convex vector optimization algorithm," *Submitted for Publication*, 2021.

[12] 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, pp. 1–24, 1998.

[13] A. Löhne, *Vector Optimization with Infimum and Supremum.* Springer, 2011.

[14] R. T. Rockafellar, *Convex Analysis.* Princeton University Press, 1970.

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

[16] A. Löhne and B. Weißing, "The vector linear program solver bensolve–notes on theoretical background," *European Journal of Operational Research*, vol. 260, no. 3, pp. 807–813, 2017.

[17] J. Jahn, *Vector Optimization - Theory, Applications, and Extensions.* Springer, 2004.

[18] T. Holzmann and J. C. Smith, "Solving discrete multi-objective optimization problems using modified augmented weighted thebychev scalarizations," *European Journal of Operational Research*, vol. 30, pp. 436–449, 2018.

[19] T. Bektaş, "Disjunctive programming for multiobjective discrete optimisation," *INFORMS Journal on Computing*, vol. 30, pp. 625–633, 2018.

[20] K. Dac̆hert and K. Klamroth, "A linear bound on the number of scalarizations needed to solve discrete tricriteria optimization problems," *Journal of Global Optimization*, vol. 61, pp. 643–676, 2015.

[21] I. CVX Research, “CVX: Matlab software for disciplined convex programming, version 2.0 beta.,” September 2012.

[22] M. Grant and S. Boyd, “Graph implementations for nonsmooth convex programs,” in *Recent Advances in Learning and Control* (V. Blondel, S. Boyd, and H. Kimura, eds.), Lecture Notes in Control and Information Sciences, pp. 95–110, Springer-Verlag Limited, 2008.

[23] J. Sturm, “Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones,” *Optimization Methods and Software*, vol. 11–12, pp. 625–653, 1999.

[24] C. Audet, J. Bigeon, and D. Cartier, “Performance indicators in multiobjective optimization,” *European Journal of Operational Research*, vol. 292, p. 397–422, 2021.

[25] S. Sayın, “Measuring the quality of discrete representationsof efficient sets in multiple objective mathematicalprogramming,” *Mathematical Programming*, vol. 87, p. 543–560, 2000.

[26] D. Ghosh and D. Chakraborty, “A direction based classical method to obtain complete pareto set of multi-criteria optimization problems,” *OPSEARCH*, vol. 52, pp. 340–366, 2015.

[27] K. Zheng, R. Yang, H. Xu, and J. Hu, “A new distribution metric for comparing pareto optimal solutions,” *Structural and Multidisciplinary Optimization*, vol. 55, pp. 53–62, 2017.

Appendix A

A.1 A Computational Study with Different Runtime Limits for Example 3

Table A.1: Computational Results under Runtime Limitation (50 seconds) Example 3

		epsilon	AUU	DLSW	KTW	UB	R	C	Adj
a=5	Error	0	0.0610	0.1428	0.0633	0.4150	0.2010	0.0925	0.0775
		0.005	0.0610	0.1159	0.0633	0.0489	0.1922	0.0472	0.0709
	HG	0	2.5348	7.4127	1.4557	5.4621	2.7797	1.1053	0.6087
		0.005	2.5348	7.2898	1.586	0.6262	1.9551	0.7466	0.6229
	Cardinality	0	38	19	43	154	156	133	156
		0.005	38	20	42	164 \44	161.8 \27.8	162 \26	161 \26
a=7	Error	0	0.0635	0.0577	0.0689	0.2163	0.5199	0.0659	0.0658
		0.005	0.0633	0.0577	0.0689	0.078	0.1298	0.0659	0.0658
	HG	0	3.6097	3.3893	2.0931	2.5031	6.977	1.0019	0.7674
		0.005	3.4895	3.3893	2.0931	0.9474	4.3267	1.0321	0.7543
	Cardinality	0	37	45	43	150	158	159	159
		0.005	38	45	43	164 \44	161.6 \26.2	157 \34	163 \28
a=10	Error	0	0.0603	0.1802	0.064	0.5282	0.184	0.1105	0.1004
		0.005	0.0603	0.1802	0.0835	0.0996	0.1225	0.139	0.1004
	HG	0	5.2053	8.6532	2.0931	-	5.8429	1.4924	1.3138
		0.005	5.2053	8.6532	3.243	1.1499	4.0538	1.5941	1.5617
	Cardinality	0	38	23	48	155	157.8	158	159
		0.005	38	23	42	160 \38	160 \28.4	154 \22	162 \30
a=20	Error	0	0.0789	0.0762	0.0945	0.3833	0.2774	0.1079	0.2199
		0.005	0.0789	0.0741	0.0945	0.123	0.2773	0.1303	0.1601
	HG	0	12.7094	8.8851	5.8671	37.5565	22.452	3.6095	4.2782
		0.005	11.3389	8.6309	5.8671	3.9389	25.2676	7.1808	5.4184
	Cardinality	0	33	45	45	136	157.4	155	157
		0.005	33	46	45	155 \34	161 \22.8	164 \38	163 \23

Table A.2: Computational Results under Runtime Limitation (100 seconds) Example 3

		epsilon	AUU	DLSW	KTW	UB	R	C	Adj
a=5	Error	0	0.0293	0.0980	0.0355	0.415	0.1433	0.0448	0.0422
		0.005	0.0319	0.0980	0.0355	0.0489	0.087	0.0447	0.0329
	HG	0	1.5539	3.3152	0.8512	5.4621	1.0544	0.4774	0.2926
		0.005	1.5924	3.3694	0.8469	0.3433	1.4441	0.3685	0.29
	Cardinality	0	59	28	71	248	283	281	280
		0.005	58	27	72	249 \72	323.2 \114.4	326 \111	319 \112
a=7	Error	0	0.0394	0.0333	0.0448	0.2163	0.2185	0.0455	0.0382
		0.005	0.0393	0.0333	0.0448	0.0286	0.0732	0.0659	0.0345
	HG	0	2.4166	1.372	2.5774	2.3386	2.042	0.51	0.4494
		0.005	2.4045	1.372	2.5774	-	1.3026	0.6218	0.3984
	Cardinality	0	48	82	68	240	285	281	287
		0.005	49	82	68	261 \81	322.6 \106.6	314 \103	323 \115
a=10	Error	0	0.0300	0.1122	0.0635	0.5282	0.3065	0.0444	0.0477
		0.005	0.0305	0.1122	0.0561	0.0584	0.0929	0.0422	0.0477
	HG	0	2.6334	7.8622	2.1896	20.5524	10.6075	0.9645	0.6971
		0.005	2.6359	7.704	1.9612	0.5731	1.0173	0.8517	0.5908
	Cardinality	0	69	29	54	254	282.8	267	283
		0.005	68	30	58	257 \77	316.8 \95.4	312 \102	323 \110
a=20	Error	0	0.0335	0.037	0.0774	0.3833	0.0985	0.0472	0.1594
		0.005	0.0339	0.037	0.0774	0.0584	0.1333	0.0931	0.0903
	HG	0	5.9792	4.6299	3.2962	37.5564	4.409	1.8988	2.0781
		0.005	5.9851	4.7261	3.3143	1.5682	5.6698	1.9479	1.3881
	Cardinality	0	67	86	77	268	312.8	277	285
		0.005	66	85	76	292 \85	312.8 \91	319\126	315 \95

Table A.3: Computational Results under Runtime Limitation (200 seconds) Example 3

		epsilon	AUU	DLSW	KTW	UB	R	C	Adj
a=5	Error	0	0.016	0.0546	0.0213	0.415	0.1089	0.0448	0.0405
		0.005	0.0151	0.0546	0.0213	0.0108	0.0405	0.0228	0.0178
	HG	0	0.6718	2.4949	0.5353	5.462	1.6491	0.3711	0.1778
		0.005	0.6417	2.3092	0.4808	0.1769	0.2699	0.2064	0.177
	Cardinality	0	109	40	122	421	464.6	474	463
		0.005	114	41	128	545 \275	650.2 \285.2	650 \279	641 \302
a=7	Error	0	0.0352	0.0201	0.0225	-	0.1281	0.0276	0.0374
		0.005	-	0.0201	0.0225	0.0148	0.0197	0.0659	0.0185
	HG	0	1.265	0.7286	0.6514	-	5.9576	0.5335	0.8885
		0.005	-	0.7329	0.667	-	0.6547	0.436	0.394
	Cardinality	0	130	154	129	-	466.6	472	469
		0.005	-	154	129	427 \277	466.6 \297.8	472 \288	469 \304
a=10	Error	0	0.0221	0.0588	0.0425	0.5282	0.0899	0.0366	0.0438
		0.005	0.0222	0.0603	0.0425	0.0154	0.0839	0.0422	0.0319
	HG	0	1.5587	4.8736	1.2532	20.5524	1.2446	0.3878	0.3796
		0.005	1.5863	5.3437	1.2717	0.3031	0.4661	0.3943	0.2962
	Cardinality	0	92	58	116	423	463	467	468
		0.005	90	57	115	541 \296	628.8 \299.6	640 \296	623 \314
a=20	Error	0	-	0.0204	0.0774	0.3832	0.0778	0.0469	0.0903
		0.005	-	0.0204	0.0774	0.0239	0.0606	0.0168	0.0476
	HG	0	-	2.0845	2.8347	32.564	4.2709	2.3304	1.6607
		0.005	-	2.0861	2.8347	0.7241	1.6708	2.0154	1.0872
	Cardinality	0	-	158	92	423	464	466	465
		0.005	-	157	92	522 \286	619 \320.4	632 \299	611 \330

A.2 Observations on Direction Selection Rule (Adj)

A.2.1 A case where $d \notin \operatorname{int} \mathbb{R}^p_+$ and $-d \notin \operatorname{int} \mathbb{R}^p_+$

Consider Example 3 where a=20. At 85th iteration where (ID) is the vertex selection rule and (Adj) is the direction parameter rule, *bensolve tools* provides 11 adjacent vertices for v. Among these vertices, 4 linearly independent vertices are chosen:

$$v = \begin{bmatrix} 0.8789 \\ -17.4185 \\ -0.8864 \\ 1.0071 \end{bmatrix}, \; A = \begin{bmatrix} 0.8945 & 0.7100 & 0.9126 & 0.8285 \\ -16.6897 & -17.0014 & -17.8247 & -17.2941 \\ -1.2960 & -0.7827 & -0.6679 & -0.8554 \\ 0.9901 & 0.9903 & 0.9520 & 0.9170 \end{bmatrix} \text{ where}$$

each column of A indicates an adjacent vertex of v. Solve the system $d = \frac{A^{-\top}e}{\|A^{-\top}e\|}$. Then we get

$$d = \begin{bmatrix} -0.8105 \\ -0.2746 \\ -0.4581 \\ 0.2405 \end{bmatrix} \text{ and } -d = \begin{bmatrix} 0.8105 \\ 0.2746 \\ 0.4581 \\ -0.2405 \end{bmatrix}.$$

A.2.2 A case where $d \in \operatorname{int} \mathbb{R}^p_+$ or $-d \in \operatorname{int} \mathbb{R}^p_+$ is guaranteed

Proposition A.2.1. *Let $p = 2$, $C = \mathbb{R}^2_+$ and $v \in V^K$ be a vertex of the outer approximation P^K. Let the adjacent vertices of v be $v^1, v^2 \in V^K$. The solution of $[v^1 \; v^2]^\top d = e$ gives $d \in \operatorname{int} \mathbb{R}^2_+$ or $-d \in \operatorname{int} \mathbb{R}^2_+$.*

Proof. We have $v, v^1, v^2 \in \operatorname{bd} \operatorname{conv}(V^K + \mathbb{R}^2_+)$. It holds that $v^1 \not< v^2$ or $v^2 \not< v^1$ as the vertices lie on the boundary of $\operatorname{conv}(V^K + \mathbb{R}^2_+)$. To see, let us assume to the contrary that $v^1 < v^2$. Since $v^1 \in \operatorname{bd} \operatorname{conv}(V^K + \mathbb{R}^2_+)$, $v^1 < v^2$ implies $v^2 \in \operatorname{int} \operatorname{conv}(V^K + \mathbb{R}^2_+)$. This contradicts that v^2 is a vertex. We can also see $v^2 \not< v^1$ similarly.

Hence, for the vertices v^1 and v^2, one of the following holds by construction: $v^1_1 \geq v^2_1$ and $v^1_2 \leq v^2_2$, or $v^1_1 \geq v^2_1$ and $v^1_2 \leq v^2_2$. Note that $v^1_1 = v^2_1$ implies $v_1 = v^1_1 = v^2_1$, since v^1 and v^2 are adjacent to v. Hence, the equality holds if and only if v, v^1 and v^2 lie on the same face. This contradicts with v being an extreme point. Hence, we have $v^1_1 > v^2_1$ and $v^1_2 < v^2_2$, or $v^2_1 > v^1_1$ and $v^2_2 < v^1_2$. Since $v^1_1 d_1 + v^1_2 d_2 = v^2_1 d_1 + v^2_2 d_2$, we have $\frac{d_1}{d_2} = \frac{v^2_2 - v^1_2}{v^1_1 - v^2_1} > 0$ which shows that d or $-d \in \operatorname{int} \mathbb{R}^2_+$. $\square$

A.3 Measuring the Quality of the Solution Set

Many performance indicators are proposed to compare solution sets of MOP found by different algorithms as the representative quality of the solution sets may differ significantly. The reader may refer to the recent survey by [24] for additional performance measures to evaluate the quality of the solution sets in MOP. Although Algorithm 1 is designed to approximate the upper image under desired proximity level, other performance indicators may also be used to compare the quality of the solution sets. Here, we describe other performance indicators besides proximity measures under the two category: distribution and spread, and cardinality. We discuss these attributes and explain how they are used to compare different variants.

A.3.1 Distribution and Spread

The solutions should be well-spread among the boundary of the upper image for the solution set to be considered representative. Moreover, it should contain C-minimal solutions from all regions of $\operatorname{bd}\mathcal{P}$.

A.3.1.1 Uniformity

The aim of obtaining uniform solutions is to minimize the redundancies in the sense that the points found on $\operatorname{bd}\mathcal{P}$ are not too close to each other. A clustered set of solutions may not be favorable to increase the representativeness of the solution sets. Sayın [25] defines uniformity as the minimum distance between any pair of the solution set, that is,

$$U = \min_{y^1 \in f(\bar{\mathcal{X}}^K)} \min_{y^2 \in f(\bar{\mathcal{X}}^K)\setminus\{y^1\}} \left\| y^1 - y^2 \right\|_2$$

The higher values of U are favorable.

A.3.1.2 Evenness

Ghosh and Chakraborty [26] define evenness as:

$$E = \frac{\max_{y^1 \in f(\bar{\mathcal{X}}^K)} \min_{y^2 \in f(\bar{\mathcal{X}}^K) \setminus \{y^1\}} \|y^1 - y^2\|_2}{\min_{y^1 \in f(\bar{\mathcal{X}}^K)} \min_{y^2 \in f(\bar{\mathcal{X}}^K) \setminus \{y^1\}} \|y^1 - y^2\|_2}$$

Clearly, $E \geq 1$ and the values that are closer to 1 indicate a better distribution of solutions.

A.3.1.3 Distribution Metric

Uniformity and evenness may cause inconsistent results when there is more than one cluster in the solution set, as they only measure the distance between a solution and its closest neighbor. Therefore the gaps between the clusters of solutions are not taken into account. To tackle these challenges distribution metric is proposed by Zheng et al. [27] for MOP which considers the standard deviation and mean of the distances between two adjacent solutions corresponding to the i^{th} objective, respectively.

Below we define another distribution metric which does not project the solution set onto each component but considers the real distances between them. For our computational experiments, we benefit from the polyhedron generated by the convex hull of the solution set, namely conv $f(\bar{\mathcal{X}}^K) + C$. We calculate the length of each edge of the polyhedron (excluding the faces generated by the extreme directions), which are the Euclidean distances between two adjacent points. The adjacency information is found by *bensolve tools*. Then we find

$$DM = \frac{\sigma}{\mu}$$

where σ and μ are the standard deviation and mean of the lengths of all edges, respectively. Instead of the method in Zheng et al. [27], it considers the real adjacency information between two solutions, rather than the adjacency of solutions reduced to one dimension.

A.3.2 Cardinality

There should be reasonable number of solutions for the solution set to be representative. Small cardinality may not be sufficient to represent the whole solution set. On the other hand, a large set of solutions may result in some regions being explored by clustered points while the other regions are represented by fewer solutions. This may cause an imbalance in coverage levels between different regions [25]. Most of the cardinality-related performance indicators presented in [24] are for discrete MOP, and it requires knowledge on the cardinality of all the non-dominated points on the Pareto Frontier. For convex MOP or, more generally CVOP, commenting on the cardinality may be more challenging. However, intuitively one seeks to approximate the solution set with the least possible number of solutions while obtaining the desired proximity level.

Figure A.1: A solution set with cardinality 8 (left) and 15 (right) for Example 1 where d=2.

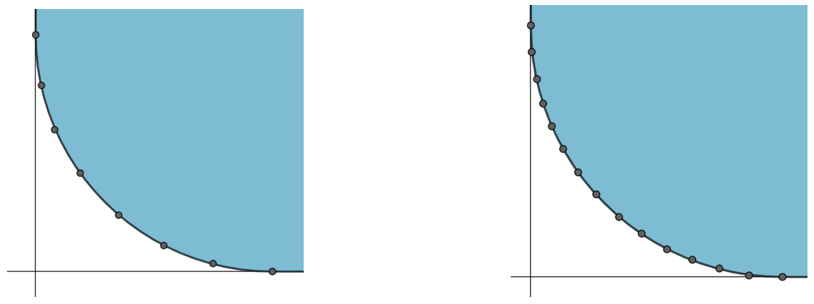


Table A.4 summarizes the performance indicators discussed above. $\Uparrow$ ($\Downarrow$) means higher (lower) values of the performance indicator are sought.

Table A.4: Desired levels of different performance indicators

Proximity	**Distribution and Spread**	**Cardinality**
Approximation Error $\Downarrow$	Uniformity $\Uparrow$	
Hypervolume Gap $\Downarrow$	Evenness $\Downarrow$	
	Distribution Metric $\Downarrow$	

Although our main indicator to compare the variants is proximity measures, we can also compare the variants with other performance indicators. Note that some of the indicators defined above cannot be used independently to compare different variants. Distribution and spread indicators may give inconsistent results (even if there are no clustered points) when the cardinalities of the solution sets differ. For instance, smaller uniformity may not necessarily suggest a decrease in representability; see Figure A.1.