Centro de Investigación y de Estudios
Avanzados del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

EDS: Método de continuación para problemas de optimización multi-objetivo mixtos-enteros

Tesis que presenta

David Laredo Razo

para obtener el Grado de

Maestro en Ciencias en Computación

Director de la Tesis

Dr. Oliver Schütze

México, D.F. **Noviembre 2015**

Centro de Investigación y de Estudios
Avanzados del Instituto Politécnico Nacional

Zacatenco Campus

Computer Science Department

EDS: A Continuation Method for Mixed-Integer Multi-objective Optimization Problems

Submitted by

David Laredo Razo

As a fulfillment of the requirement for the degree of

Master in Computer Science

Advisor

Dr. Oliver Schütze

México, D.F. **November 2015**

Resumen

Los problemas de optimización multi-objetivo (MOPs por sus siglas en inglés) se presentan frecuentemente en diversas aplicaciones de ingeniería y gestión. Un tipo más general de MOPs son los llamados problemas de optimización multi-objetivo mixtos (MMOPs en inglés), en los cuales el espacio de búsqueda está conformado por variables enteras y variables reales. La investigación relacionada con los MMOPs es aún escasa considerando la frecuencia con la que estos se presentan en la vida real. Por ejemplo, en aplicaciones gestión, el problema de la distribución óptima de los recursos en una línea de ensamblaje puede involucrar variables enteras (número de trabajadores, número de estaciones de trabajo, número de piezas de un determinado producto, etc.), así como variables reales (tiempo consumido por tarea, costo de producción. etc.). Generalmente, la resolución de MMOPs es más complicada que la de los MOPs, requiriendo de técnicas especializadas.

En este trabajo presentamos un método de continuación para la resolución tanto de MOPs como de MMOPs. Nuestro algoritmo, llamado *Enhanced Directed Search (EDS)*, permite dirigir la búsqueda en una dirección predefinida en el espacio objetivo. Más aún, este método puede aplicarse para la resolución de problemas con $k \geq 2$, donde k denota el número de objetivos que comprenden al problema, a la vez que hace uso de puntos ya calculados en el vecindario para aumentar la eficiencia del método. Demostramos, mediante la resolución de algunas funciones de prueba, que el método EDS es tan confiable como los métodos Direct Zig Zag (DZZ) y NSGA-II pero más eficiente que cualquiera de ellos.

Abstract

Multi-objective optimization problems (MOPs) commonly arise in several real-life engineering and management applications. A more general (and hence more complicated to solve) kind of MOPs are the so called mixed-integer multi-objective optimization problems (MMOPs), where the search space is composed by real and integer variables. The research regarding the treatment of MMOPs is still scarce despite the frequency in which they arise in practice. For example, in management applications, the allocation of resources in a factory may consider integer variables (number of workers, number of workstations, number of pieces of a determined product, etc.) and real variables (time consumed per task, production cost, etc.). These MMOPs are generally more complicated to solve, and thus, require specialized techniques.

In this work we propose a continuation method that deals with continuous and mixed-integer problems as well. Our approach, called *Enhanced Directed Search (EDS)*, is capable of steering the search along a predefined direction in objective space. Furthermore, it is capable of dealing with problems with $k \geq 2$, where k denotes the number of objectives involved in the problem, while making use of neighboring information to increase the efficiency of the method. We demonstrate, through the resolution of some test functions, that the EDS method is as reliable as the Direct Zig Zag (DZZs) and NSGA-II methods while being more efficient than both of them.

Agradecimientos

A mi Dios Jehova por sus abundantes bendiciones, por permitirme concluir este grado académico y por ser siempre un pilar fundamental en mi desarrollo humano.

A mi asesor, el Dr. Oliver Schütze, sin cuya guía y constantes críticas y observaciones este trabajo no habría sido posible. Por su confianza y el tiempo invertido en el desarrollo de este proyecto así como por el amable trato recibido. Le agradezco por el conocimiento compartido y por sus magistrales clases, las cuales me motivaron al estudio de esta ciencia y a la realización de este proyecto de tesis. Por su buena disposición para la conclusión exitosa de este trabajo y por ser siempre una agradable persona.

A mis padres Angélica y Rogelio a los cuales amo profundamente y que siempre han sido un apoyo incondicional e invaluable, no solo en mi vida profesional y académica, sino en todos los aspectos de mi vida. Ellos merecen gran parte del mérito de este trabajo. A mi hermano Hiram, al cual amo por igual y que siempre ha sido un amigo entrañable del cual he podido aprender.

A mis amigos Jhonatan y Oliver, por estar en los buenos y en los malos momentos, por su disposición a ayudar y por ser siempre un apoyo incondicional durante este proceso. A mis amigos Martín, Gabriel y Cesar por hacer de la maestría una experiencia amena y gratificante. A todos ellos por ayudarme a ser mejor persona.

Al Dr. Honggang Wang por haberme brindado tan amable recibimiento durante mi estancia en el extranjero, por haber contribuido al desarrollo de este trabajo y por ser una excelente persona con una gran calidad humana.

Al CINVESTAV-IPN por haber sido mas que mi escuela, mi casa durante estos dos años, por las facilidades que me brindó para mis estudios de posgrado y al CONACyT por la beca de maestría y el apoyo brindado para mi estancia en el extranjero.

A mis sinodales los doctores Luis Gerardo de la Fraga y Amilcar Meneses, por haber aceptado revisar y hacer observaciones a este trabajo. A los doctores Francisco Rodríguez Henríquez y Debrup Chakraborty por sus clases, el conocimiento compartido y la humildad con la que siempre se han conducido.

A todo el personal del Departamento de Computación por las facilidades brindadas durante estos dos años.

Gracias a todos los que de una u otra manera formaron parte directa o indirectamente de este proceso.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

BOP Bi-objective Optimization Problem. 14, 20, 37

DS Directed Search. 3–5, 16, 18, 19, 27, 28, 31, 43, 71

DZZ Direct Zig Zag. v, vii, 5, 19, 20, 53–68, 72, 77

EDS Enhanced Directed Search. v, vii, 4, 18, 27–29, 34, 36, 37, 40–42, 45, 47, 49–51, 53–69, 71–73, 77, 78

FPS First Pareto Solution. 20, 53

GD Generational Distance. 22–24

GSA Gradient Subspace Approximation. 4, 43

IFT Implicit Function Theorem. 12, 13

IGD Inverted Generational Distance. 22–24

KKT Karush-Kuhn-Tucker. 7–9, 11, 13, 16

MMOP Mixed-Integer Multi-objective Optimization Problem. v, vii, 2–5, 10, 21, 27, 45, 47, 71

MOEA Multi-objective Optimization Evolutionary Algorithm. 5, 20, 21, 43, 73

MOEA-D Multi-objective Optimization Evolutionary Algorithm based on Decomposition. 21

MOO Multi-objective Optimization. 1, 4, 5

MOP Multi-objective Optimization Problem. v, vii, 1, 2, 5–7, 10–15, 19, 21–24, 27, 37, 45, 46, 53, 54

NBI Normal Boundary Intersection. 16

NSGA-II Non-Sorted Genetic Algorithm II. v, vii, 21, 53–69, 77, 78

PC Predictor-Corrector. 3, 4, 11–13, 16–18, 27, 37, 41

PF Pareto Front. 48

SOP Single-objective Optimization Problem. 33

Chapter 1

Introduction

Optimization is a research field that deals with the problem of obtaining the *best* attainable solution of a problem, with regard to some criteria, from some set of available alternatives. We can find applications for the optimization in many different fields, for instance, investors seek to create portfolios that avoid excessive risk while achieving a high rate of return, manufacturers aim for maximimum efficiency in the design and operation of their productions processes, engineers adjust parameters to optimize the performance of their designs, etc.

Optimization is an important tool in several fields, however, to make use of this tool we must first identify some *objective*, i.e., a quantitative measure of the performance of the system under study. This objective could be profit, time, quality, performance or any quantity or combination of quantities that can be represented by a *single* value. The objective depends on certain characteristics of the system, called *variables* and sometimes, just as in real life, the variables of a problem are somehow restricted or *constrained*. The goal is then to find the set of variables that optimizes the objective value, this is what we call *single objective optimization*.

A less intuitive scenario, nevertheless present in a great variety of applications, arises when several objectives have to be optimized *concurrently* under the aforementioned conditions. This is what we call *multi-objective optimization (MOO)*. As a simple example consider product manufacturing: the product quality has to be maximized while the production cost has to be minimized. A high quality product is likely to increase the production costs, which translates into less profits for the producers. On the other hand, cheap products may lead to costumer insatisfaction, which will probably transform into low product demand. This is an example of a *Multi-objective Optimization Problem (MOP)*.

It is therefore mandatory to achieve a consensus over the meaning of an optimal solution according to more than one objective. For this purpose we use the notion of

Pareto optimality [1], which states that a solution is defined to be better if at least one of its objectives is improved while the others do not get worse. It is worth to mention that typically there is no single solution to multi-objective problems. Instead, a whole set of solutions can be found where each solution in the set represents a different optimal solution. Hence, an optimal solution is one which is not dominated by any other one within the feasible set. Mathematical definitions of this concept shall be reviewed in the next chapters.

One important sub-field in optimization is the treatment of the so called mixed-integer problems[1] given that they frequently arise in practice. For example in management sciences, problems such as supplier selection [2, 3, 4] or assembly line balancing [5, 6] may consider integer variables (number of workstations, number of suppliers) as well as continuous variables (percentage of demmand assigned to each supplier, cost per worker). These discrete or mixed-integer multi-objective problems require specialized techniques for their numerical solutions. The goal of this thesis is the development of a state-of-the-art method for solving such mixed-integer multi-objective problems (MMOPs).

1.1 Motivation

Although there is a great amount of work related to the solution, structure and even creation of multi-objective problems, most of it deals with continuous problems[2]. Several strategies for the treatment of continuous MOPs have been proposed and tested over the past decades. For example, scalarization methods [7, 8, 9, 10, 11] that transform (via parametrization) multi-objective problems into single objective ones. By choosing a clever sequence of parameters, a suitable finite size approximation of the Pareto set can be obtained. Further, there exist many set oriented methods such as evolutionary strategies [12, 13, 14, 15], subdivision techniques [16, 17] or cell mappig techniques [18, 19]. These methods have in common that they use sets in an iterative manner and are thus able to deliver an approximation of the solution set in one run. Nevertheless, they often require a high number of function evaluations, making efficiency an issue. Another group encloses the descent direction methods [20, 21, 21, 22] that are generally fast local convergent algorithms focused on finding only one optimal point. Finally, we can find the so called continuation methods [23, 24, 25, 22, 26, 27] which exploit the fact that, under some mild conditions, the Pareto set/front forms typically a $(k-1)$-dimensional object [27], making it possible to apply these techniques to detect further points within a neighborhood of a given

[1] In mixed-integer problems some variables are real and some are integer. An in depth definition of mixed-integer problems will be given in Chapter 2.

[2] By continuous optimization we mean that all the functions involved (objectives as well as constraints) are continuous.

solution. Literally, the idea here is to follow the curve (or manifold[3]) of Pareto points which turns out to be a very efficient manner to detect large connected portions of the solution set. These methods are of local nature, implying that we can get trapped in local optima or miss sections of the Pareto set if it is not connected.

Despite the amount of work developed for continuous MOPs the research done on discrete or mixed-integer problems is still scarce (to the best of our knowledge) if we take into account the frequency in which this problems arise in real life, specially in engineering and management problems [29, 30, 31, 32]. The important gap in this field and that novel and efficient methods can be developed in this area was our main motivation for this research.

1.2 The Problem

This thesis project deals with the development of a new continuation method to approximate the Pareto set of a given MMOP while making it competitive against the state-of-the-art methods in this field. More specifically, given a MMOP and an initial Pareto point of the problem, the task is to explore the neighborhood of such initial point in order to find further Pareto optimal solutions. The method will continue in an iterative manner until a suitable discretization of the Pareto set/front is computed. By a *suitable discretization* we mean a set of points that are evenly distributed along the objective space. The theory developed for this method is of local nature and hence we consider that the starting point is *close enough* to the set of interest. Given the local nature of the method we do not expect to obtain the entire Pareto set/front, but instead some of the connected components of it. Constraint handling techniques, as well as convergence proofs are left out of this work to be subject of a future analysis.

1.3 Aims of the Thesis

Our main goal in this project is the development of a state-of-the-art *continuation method* for solving mixed-integer multi-objective optimization problems (MMOPs). The method's efficiency will be measured in terms of function evaluations, while its performance will be measured in terms of the $\triangle_p$ (*Delta p*) performance indicator [33]. Specifically, we aim to design a *Predictor-Corrector (PC)* method for the numerical treatment of MMOPs. To achieve this goal the Directed Search (DS) method [22] will serve as the basis for the development of our new algorithm.

[3]A manifold is a topological space that resembles Euclidean space near each point. More precisely, each point of a n-dimensional manifold has a neighborhood that is homeomorphic to the Euclidean space of dimension n [28].

The Enhanced Directed Search (EDS) method, as we call our approach, will improve the steering properties of the original DS method (which has some drawbacks regarding the detection of Pareto points). For this, new mathematical theory for mapping movements in objective space into decision space will be tested. Efficiency plays also an important role in the development of the EDS method. In order to make the EDS method highly efficient the approach taken in the Gradient Subspace Approximation (GSA) method [34], which considers exploiting neighborhood information in order to save function evaluations, will be considered. The EDS method should also be applicable to problems with more than two objectives, which can be easily accomplished, thanks to the method's steering properties, by implementing the data structure used in [35]. Finally, some modifications will be done to the method in order to make it capable for solving MMOPs.

1.4 Final Contributions

This thesis contributes to the area of multi-objective optimization, specifically it introduces a new PC method called *Enhanced Directed Search (EDS)* capable of following a path along the Pareto front of MMOPs. First order gradient information is used, nevertheless neighborhood information is used as well improving the overall efficiency of the method. Problems with more than two objectives can be solved with the EDS method. Finally, the method along with the theory developed in this thesis is also applicable to continuous multi-objective optimization. In this case our method has shown to perform as good as state-of-the-art methods.

1.5 Organization of the Thesis

The remainder of this thesis is organized as follows: Chapter 2 gives the theoretical background of MOO as well as a brief description of the state-of-the-art methods for solving MMOPs, an introduction to the DS method is given in this chapter as well. The EDS method is described in detail in Chapter 3, beginning with the theoretical foundation of the method, a whole description of the algorithm and its components is then given. Results on several benchmark functions are shown in Chapter 4 together with a comparison against state-of-the-art methods for solving MMOPs. Finally, Chapter 5 presents our conclusions about the method along with some ideas to be considered for future work.

Chapter 2

Background and Related Work

This chapter is dedicated to the exposition of the theoretical background of multi-objective optimization (MOO), needed to understand the ideas developed in this work. First, a description of MOO is given followed by the definition of a mixed-integer multi-objective problem (MMOP). Next, an introduction to continuation methods is given, Directed Search (DS) and Direct Zig-Zag (DZZ) methods are reviewed in detail. A brief description of multi-objective optimization evolutionary algorithms (MOEAs) follows. Finally, an introduction to performance indicators and the $\triangle_p$ indicator is given.

2.1 Theoretical Background

2.1.1 Multi-objective Optimization

Optimization is a field where one tries to find solutions that represent the best choice of a given problem. Here, we divide this field into two sub-fields: (i) single objective optimization, where one tries to solve problems with only one objective and (ii) when a problem has multiple objectives which are in conflict with each other and all have to be optimized concurrently. This latter are called multi-objective optimization problems (MOPs).

Arguably, optimization plays an important role in everyday life, for example in activities such as decision-making, design of goods, system's control and others [36]. Hence, it is important to improve the knowledge related to this field and specifically application domains that involve MOPs, for example: engineering applications (electrical, aeronautical, robotics and control), industrial applications (scheduling, management, design and manufacture) or scientific applications (chemistry, physics,

medicine and computer science) [37].

As a simple example imagine the process of building a motorcycle. Such a vehicle has many features, but for now let us focus on just two of them: the speed and the cost. We want our model to be attractive to the costumers due to the quality, but also to get a good profit selling it. Both objectives, cost and quality, are in conflict. Shall we reduce the production cost, the quality might be affected. If on the other hand, we maximize the quality, the motorcycle might be too expensive for the costumers. This implies that not only one solution exists but rather an entire set of solutions. This set is called the Pareto set [1] in the literature.

Formally speaking, a continuous MOP is defined as follows:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & F(x) \\ \text{s. t.} \quad & \\ & h_i(x) = 0, \ i = 1 \ldots m \\ & h_j(x) \leq 0, \ j = 1 \ldots p, \end{aligned} \tag{2.1}$$

where $F : \mathbb{R}^n \to \mathbb{R}^k$, $F(x) = (f_1(x), \ldots, f_k(x))^T$ represents a vector of $k \geq 2$ *objective functions*. The *feasible decision vectors*, that form the set $\mathbb{X}$, are those $x \in \mathbb{R}^n$ that comply with the equality $h_i(x)$ and inequality $h_j(x)$ constraints.

The optimality of a MOP is defined by the concept of dominance [1].

Definition 2.1.1 (Pareto Dominance) *A point $y \in \mathbb{X}$ is dominated by a point $x \in \mathbb{X}$ ($x \prec y$) with respect to Eq.* (2.1) *if x is partially less than y, i.e., if $f_i(x) \leq f_i(y)$, for all $i \in 1, \ldots, k$, and $f_j(x) < f_j(y)$ for some $j \in 1, \ldots, k$. Otherwise it is non-dominated by x.*

Definition 2.1.2 (Pareto Optimality) *A decision vector $x^* \in \mathbb{X}$ is Pareto optimal with respect to Eq.* (2.1) *if there does not exist another decision vector $x \in \mathbb{X}$ such that $x \prec x^*$.*

Definition 2.1.3 (Pareto Weak Optimality) *A decision vector $x^* \in \mathbb{X}$ is* weakly *Pareto optimal if there does not exist another decision vector $x \in \mathbb{X}$, such that $f_i(x) < f_i(x^*) \quad \forall i = 1, ..., k$.*

In general, the solution of a MOP consists not only of a single solution but of a set of solutions which have to be considered as optimal. The solution set is called the *Pareto set* and its corresponding image is called the *Pareto front* which typically forms a $(k-1)$-dimensional manifold [27], where k is the number of objectives involved

in the problem. The concepts of Pareto set and Pareto front are formalized in the following definition:

Definition 2.1.4 (Pareto set and Pareto front) *The set of optimal points $\mathcal{P}$ for Eq.* (2.1)*,*

$$\mathcal{P} = \{x \in \mathbb{X} \mid \nexists y \in \mathbb{X} : y \prec x\}$$

is called the Pareto set. The image $F(\mathcal{P})$ is called the Pareto front.

Figure 2.1 shows a hypothetical Pareto front for the cost-quality problem. A and B are non-dominated points while C is a dominated one.

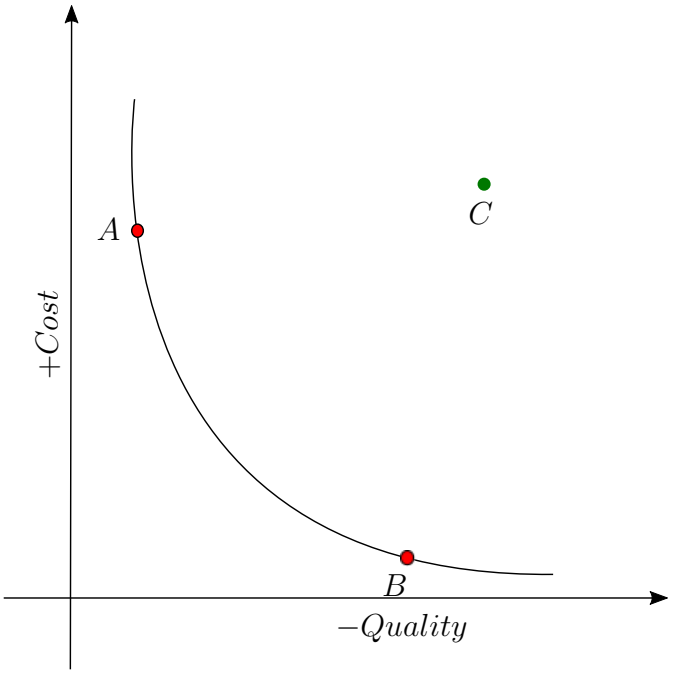


Figure 2.1: Hypothetical Pareto front for the cost-quality problem

2.1.2 The Karush-Kuhn-Tucker Optimality Conditions for Multi-objective Optimization

Simultaneously to their single objective optimization first order conditions (also known as Karush-Kuhn-Tucker or KKT conditions) [38] Karush, Kuhn and Tucker formulated a first order necessary condition for MOPs.

Theorem 1 (KKT Conditions) *Let x^* be a solution of Eq. (2.1), that is a Pareto point, then there exist vectors $\alpha \in \mathbb{R}^k$ and $\lambda \in \mathbb{R}^{m+p}$ s.t.*

$$\alpha_i \geq 0 \text{ and } \sum_{i=1}^{k} \alpha_i = 1$$

and

$$\begin{aligned} \sum_{i=1}^{k} \alpha_i \nabla f_i(x^*) + \sum_{j=1}^{m+p} \lambda_j \nabla h_j(x^*) &= 0 \\ h_i(x^*) &= 0, \qquad i = 1 \dots m \\ \lambda_j \geq 0, \quad h_j(x^*) \leq 0, \quad \lambda_j h_j(x^*) &= 0, \qquad j = 1 \dots p. \end{aligned} \tag{2.2}$$

By introducing the scalar valued function $g_\alpha : \mathbb{R}^n \to \mathbb{R}$,

$$g_\alpha = \sum_{i=1}^{k} \alpha_i f_i(x^*), \tag{2.3}$$

it can be noticed that $\nabla g_\alpha(x^*) = \sum_{i=1}^{k} \alpha_i \nabla f_i(x^*)$. It can be seen then, that Eq. (2.3) is equivalent to the claim that x^* is a KKT point of the corresponding scalar-valued (single objective) optimization problem with the objective function g_α [36].

Due to Eqs. (2.2) and (2.3), g_α constitutes a convex linear combination of the individual objective functions f_i, where each coefficient indicates the relative weight with which the individual objective f_i is part of the linear combination g_α.

One important thing to outline is that given a KKT point x^* its associated weight vector α is normal to the linearization (tangent) of the Pareto front at $F(x^*)$ [27] as it is illustrated in Figure 2.2.

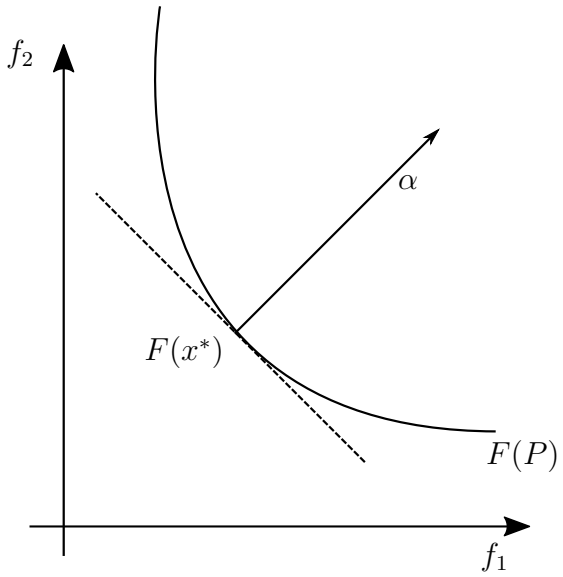


Figure 2.2: The weight vector α is orthogonal to the linearized Pareto front $F(P)$ at $F(x^*)$.

The Jacobian of F at a point x is given by

$$J(x) = \begin{pmatrix} \nabla f_1(x) \\ \vdots \\ \nabla f_m(x) \end{pmatrix} \in \mathbb{R}^k. \tag{2.4}$$

By considering the unconstrained version of Eq. (2.1) Theorem 1 can be restated as:

Theorem 2 *Let x^* be a local Pareto point of an unconstrained multi-objective problem with $y^* = F(x^*)$. Let g_α denote a convex combination of the objectives for which x^* is a KKT point.*

Then the weight vector α is a kernel vector of the Jacobian of F at x^*, that is, $J(x^*)^T\alpha = 0$.

As a corollary, it is obtained that $\text{rank}(J(x^*)^T) < k$ for any x that is a KKT point [27].

Finally, an important aspect for the context of our work is that Pareto points typically form a $(k-1)$-dimensional differentiable manifold [27]. This and subsequent applications will be subject to an in-depth discussion throughout the thesis.

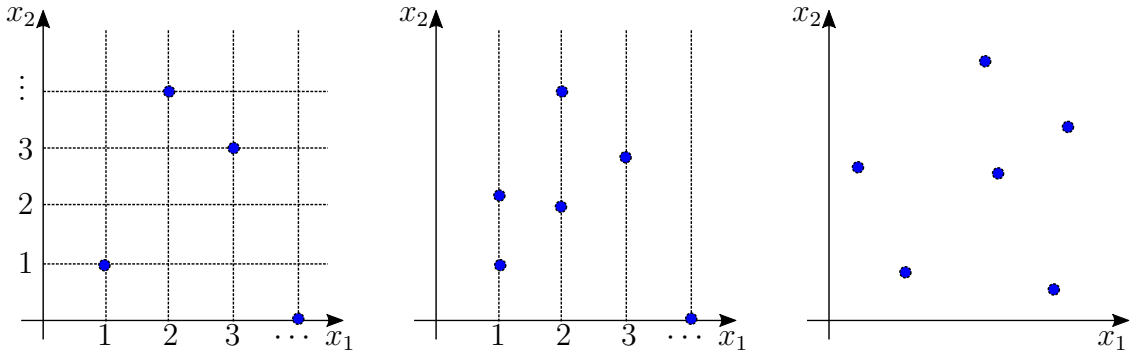
2.1.3 Mixed-Integer Multi-objective Optimization

A mixed-integer multi-objective optimization problem (MMOP) can be formally stated as:

$$\begin{aligned} \min_{x \in \mathbb{Z}^{d_1} \times \mathbb{R}^{d_2}} \quad & F(x) \\ \text{s. t.} \quad & \\ & h_i(x) = 0, \ i = 1 \dots m \\ & h_j(x) \leq 0, \ j = 1 \dots p, \end{aligned} \tag{2.5}$$

where $F : \mathbb{Z}^{d_1} \times \mathbb{R}^{d_2} \to \mathbb{R}^k$, which means that the parameter vector can be formed either by real variables, discrete (or integer) variables or a mixture of both depending on the values of d_1 and d_2 respectively. For instance, Figure 2.3a displays a two dimensional integer space where $x_1, x_2 \in \mathbb{Z}$, Figure 2.3b represents a mixed-integer space where $x_1 \in \mathbb{Z}$ and $x_2 \in \mathbb{R}$, while in Figure 2.3c $x_1, x_2 \in \mathbb{R}$, that is, they belong to a real space. Just as in the case of continuous MOPs h_i and h_j represent the constraints of the problem. The feasible set $\mathbb{X}$ is the set formed by all the $x \in \mathbb{R}^n$ that comply with the constraints.

The goal of Eq. (2.5), as in the continuous case, is to seek non-dominated (Pareto optimal) solutions of the objective function F on the feasible set $\mathbb{X}$.



(a) Integer space (b) Mixed-Integer space (c) Real space

Figure 2.3: Examples of decision spaces

In this work the theory developed for continuous MOPs will be used for the treatment of MMOPs since it can be shown, that under some assumptions, most of theory presented within this chapter holds for MMOPs.

2.2 Continuation Methods

The main motivation to apply continuation methods for the numerical treatment of MOPs comes from the fact that, under certain mild assumptions, it can be induced from the KKTs conditions that the Pareto front of a continuous MOP is typically a $(k-1)$-dimensional manifold [27], where k is the number of objectives. Thus, specialized techniques capable of performing a search along the manifold of solutions are very promising within this context.

Continuation (also known as embedding or homotopy) methods can be seen as methods for following a certain curve or manifold [39]. The classical embedding methods may be regarded as a forerunner of Predictor-Corrector (PC) methods. Stated briefly, a homothopy method consists of the following: suppose one wishes to obtain a solution to a system of n nonlinear equations in n variables, say

$$F(x) = 0, \tag{2.6}$$

where $F : \mathbb{R}^n \to \mathbb{R}^n$ is a sufficiently smooth mapping[1]. Let us consider the situation in which very little *a priori* knowledge concerning zero points of F is available. Certainly, if on the contrary a good approximation x_0 of zero point x^* of F is available, it is advisable to calculate x^* via a Newton type algorithm [20].

Since it is assumed that *a priori* knowledge is not available, a possible remedy is to define a homotophy or deformation $H : \mathbb{R}^{n+1} \to \mathbb{R}^n$ such that

$$H(x, 1) = G(x), \quad H(x, 0) = F(x), \tag{2.7}$$

where $G : \mathbb{R}^n \to \mathbb{R}^n$ is a (trivial) smooth map having known zero points. Typically, one may choose a *convex homothopy* such as:

$$H(x, \lambda) = \lambda G(x) + (1 - \lambda)F(x) \tag{2.8}$$

and attempt to trace an implicitly defined curve $c(s) \in H^{-1}(0)$ from a starting point $(x_1, 1)$ to solution point $(x^*, 0)$. If this succeds then a zero point x^* of F is obtained.

We can now describe the basic ideas of continuation methods. Assume we are given the equation:

$$H(x, \lambda) = 0, \tag{2.9}$$

[1]When we say a map is sufficiently smooth we mean that it has as many continuous derivatives as the subsequent discussion requires.

where $H : \mathbb{R}^{n+1} \to \mathbb{R}^n$ is sufficiently smooth. If (x, λ) is a solution of Eq. (2.9) with $\text{rank}(H'(x, \lambda)) = n$, then it follows by the Implicit Function Theorem (IFT) [27] that there exists a value $\epsilon > 0$ and a curve $c : (-\epsilon, \epsilon) \to \mathbb{R}^{n+1}$, such that $c(0) = x$ and

$$H(c(s)) = 0, \quad \forall s \in (-\epsilon, \epsilon). \tag{2.10}$$

By differentiating Eq. (2.10) we get

$$H'(c(s))^T \cdot c'(s) = 0. \tag{2.11}$$

Hence, tangent vectors $c'(s)$ (and thus, linearizations of the solution curve at $x = c(s)$) can be found by computing kernel vectors of $H'(x, \lambda)$. This is done in literature by a QR factorization of $H'(x, \lambda)^T$ [39] : if

$$H'(x, \lambda)^T = QR = (q1, \ldots, q_{n+1})R \tag{2.12}$$

for an orthogonal matrix $Q \in \mathbb{R}^{(n+1)\times(n+1)}$ and a right upper triangular matrix $R \in \mathbb{R}^{(n+1)\times n}$, then the last column vector q_{n+1} of Q is such a desired kernel vector. The orientation of the curve (note that both $+q_{n+1}$ and $-q_{n+1}$ are desired linearizations, but point in opposite directions) can be inferred by monitoring the sign of

$$\det\begin{pmatrix} H'(x, \lambda) \\ q_{n+1}^T \end{pmatrix}. \tag{2.13}$$

More specifically, we change the sign of the computed predictor whenever the determinant in Eq. (2.13) is negative. Having the vector pointing along the linearized solution curve, a movement in that direction can be performed leading to a predictor point p. In a following corrector step one can get back to c by using Eq. (2.9), e.g., via a Gauss-Newton or a Levenberg-Marquardt method [40] starting with p. In this manner, a sequence of solutions that are aligned along the curve $H^{-1}(0)$ can be obtained. Figure 2.4 provides a more illustrative example of the working principle of PC methods.

The relation between the classical continuation theory and multi-objective optimization is thus easily derived: the first-order optimality conditions for MOPs (see Theorem 1) lead to an undetermined system of equations (that we call indistinctly KKT system or KKT equations). Thus, applying the IFT on the underlying system and under some mild conditions, it is inferred that the optimal solution set is a $(k-1)$-dimensional object that can be tracked, e.g., by PC methods. A more detailed argument on this subject is provided in [27]. The algorithms we review next closely

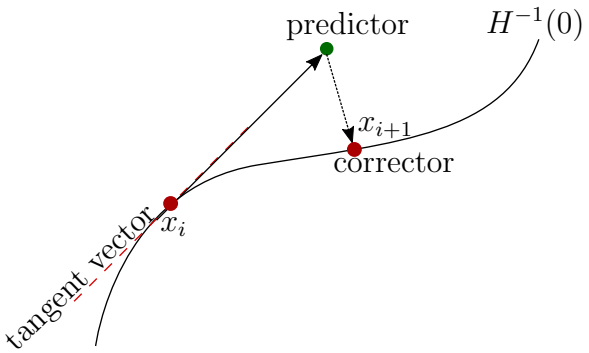


Figure 2.4: Example of a PC step.

follow the spirit of the continuation theory (in particular, PC methods) although not all are oriented to explicitly solve the KKT system of equations.

2.2.1 Hillermeier's Predictor-Corrector Method

A direct application of PC methods on the KKT equations of a MOP was first introduced by Claus Hillermeier in [27] which we describe in the following. Consider the map:

$$\min_{x \in \mathbb{R}^n} F(x), \tag{2.14}$$

where F is defined as in Eq. (2.1).

Now we can define the map $\hat{F} : \mathbb{R}^{n+k} \to \mathbb{R}^{n+1}$,

$$\hat{F}(x, \alpha) = \begin{pmatrix} \sum_{i=1}^{k} \alpha_i \nabla f_i(x) \\ \sum_{i=1}^{k} \alpha_i - 1 \end{pmatrix} = 0, \tag{2.15}$$

were $\alpha \in \mathbb{R}^k$ is the associated weight vector to $F(x)$. The set of KKT points of Eq. (2.14) is contained in the zero set of $\hat{F}$. For bi-objective optimization problems (BOPs), the method proceeds as the general homothopy technique described in Section 2.2 but with the following change: instead of computing the determinant given in Eq. (2.13) to orientate the direction of the predictor, the author proposes to check

whether the condition

$$[\mathbf{x} - \tilde{\mathbf{x}}] \cdot q \geq 0 \tag{2.16}$$

is met, where $\mathbf{x} = (x, \alpha) \in \mathbb{R}^{n+k}$ is the current corrector point, $\tilde{\mathbf{x}}$ is the previously solution, and q is the tangent vector. If that is not the case, the direction of q is flipped. Then, a suitable step length that guarantees a uniform spread of solutions on the front is sought. That is, for two consecutive solutions $\mathbf{x}$ and $\tilde{\mathbf{x}}$, it is desirable that

$$||F(\mathbf{x}) - F(\tilde{\mathbf{x}})|| \approx \tau, \tag{2.17}$$

where $\tau > 0$ is a user specified value. For this, one can take the step size

$$t = \frac{\tau}{||J(x)q||}. \tag{2.18}$$

The case were $k \geq 2$ is handled by taking kernel vectors of $\hat{F'}^T$. Given that

$$\hat{F'}(x, \alpha)^T = QR = (q_1, \ldots, q_{n+k})R \tag{2.19}$$

for an orthogonal matrix $Q \in \mathbb{R}^{(n+k)\times(n+k)}$ and a right upper triangular matrix $R \in \mathbb{R}^{(n+k)\times 1}$, the last $k-1$ column vectors of Q form an orthonormal basis of the linearized solution set in the compound (x, α)-space. Thus, one can e.g., move in the directions of the computed orthonormal vectors $x_{n+2}, \ldots, x_{x+k}$ to obtain predictors that are grid-aligned along the tangent space of the optimal manifold in decision space. A problem of this election, though, is that after mapping the computed predictors to the objective space, the grid alignment is probably not kept. Finally, the continuation algorithm is stopped if one of the Lagrange multipliers α_j, $j \in \{1, \ldots, k\}$ is negative, which indicates that a non-optimal solution has been found.

2.3 The Directed Search Method

This method defines a way to steer the search for continuous MOPs by using a direction in objective space and mapping it into parameter space [22]. The main idea is as follows.

Assume a point $x_0 \in \mathbb{R}^n$, in parameter space, with $\text{rank}(J(x_0)) = k$, and a vector $d \in \mathbb{R}^k$ representing a desired search direction in image space are given. Then, a

search direction $\nu \in \mathbb{R}^n$ in decision space is sought such that for $y_0 := x_0 + t\nu$, where $t \in \mathbb{R}_+$ is the step size (i.e., y_0 represents a movement from x_0 in direction v), it holds:

$$\lim_{t \to 0} \frac{f_i(y_0) - f_i(x_0)}{t} = \langle \nabla f_i(x_0), \nu \rangle = d_i, \qquad i = 1, ..., k. \tag{2.20}$$

Using the Jacobian of F, Eq. (2.20) can be stated in matrix vector notation as

$$J(x_0)\nu = d. \tag{2.21}$$

Hence, such a search direction ν can be computed by solving a system of linear equations. Since typically the number of decision variables is (much) higher than the number of objectives for a given MOP, i.e., $n \gg k$, system (2.21) is (probably highly) underdetermined, which implies that its solution is not unique. One possible choice is to take

$$\nu_+ = J(x_0)^+ d, \tag{2.22}$$

where $J(x_0)^+ \in \mathbb{R}^{n \times k}$ denotes the pseudo inverse[2] of $J(x_0)$. A new iterate x_1 can be computed as the following discussion shows: given a candidate solution x_0, a new solution is obtained via $x_1 = x_0 + t\nu$, where $t > 0$ is a step size and $\nu \in \mathbb{R}^n$ is a vector that satisfies (2.21). Among the solutions of system (2.21), ν_+ is the one with the smallest Euclidean norm. Hence, given t, one expects for a step in direction ν_+ (decision space) the largest progress in d-direction (objective space).

Using the ideas mentioned above, the authors of [22] developed a method to move towards and along the Pareto front of continuous MOPs which we will describe in the following.

2.3.1 Directed Search Descent Method

Given a direction $d \in \mathbb{R}^k \backslash \{0\}$ with $d_i \leq 0, i = 1, \ldots, k$, a point $x_0 \in \mathbb{R}^n$ with $\text{rank}(J(x_0)) = k$ and assuming that the image of F is bounded from below, a greedy search in direction d using Eq. (2.22) leads to the (numerical) solution of the following initial value problem:

$$\begin{aligned} x(0) &= x_0 \in \mathbb{R}^n \\ \dot{x}(t) &= J(x(t))^+ d, \quad t > 0. \end{aligned} \tag{2.23}$$

[2] If the rank of $J := J(x_0)$ is k (i.e., maximal) the pseudo inverse is given by $J^+ = J^T(JJ^T)^{-1}$.

Definition 2.3.1 (Critical Point) *Let $\gamma : [0, t_f] \to \mathbb{R}^n$ be a solution of Eq. (2.23) and let t_c be the smallest value of $t > 0$ such that*

$$\nexists v \in \mathbb{R}^n : \quad J(x(t))v = d. \tag{2.24}$$

.

Then t_c and $\gamma(t_c)$ are a critical value an critical point of (2.23) respectively.

By Definition 2.3.1, it is possible to divide the solution $\gamma : [0, t_f]$ of Eq. (2.23) into two phases, (see Figure 2.5):

- $\gamma([0, t_c])$: the function $F(\gamma(t))$ gets the desired decay in d-direction.
- $\gamma((t_c, t_f])$: the function $F(\gamma(t))$ moves along the critical points of F. For the end point $\gamma(t_f)$, it holds $J(\gamma(t_f))^+ d = 0$.

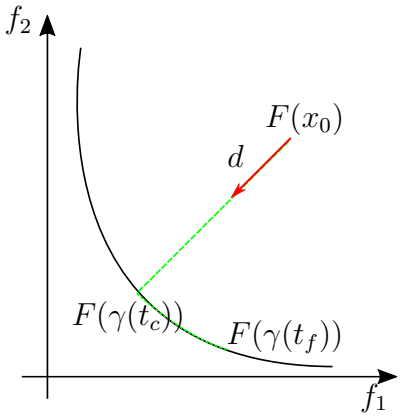


Figure 2.5: Greedy movement of DS descent

The study made in [22] shows that *critical points* are not necessarily KKT points but are local solutions of the well known NBI [11] problem, therefore the DS descent method is only restricted to the detection of such critical points. To trace the solution curve of Eq. (2.23) numerically, specialized PC methods [39] can be used. It is important to notice that by performing a movement along d-direction then, for every point x on the solution curve, it holds

$$F(x) = F(x_0) + \lambda_y d, \tag{2.25}$$

where $\lambda_y \in \mathbb{R}^+$. Hence, the curve is contained in the zero set of

$$H : \mathbb{R}^{n+1} \to \mathbb{R}^k, \quad H(x, \lambda_y) = F(x) - F(x_0) - \lambda_y d. \tag{2.26}$$

To comply with the needs of PC methods an additional parameter λ_x is introduced into the solution curve (which is defined in decision space):

$$\begin{aligned} x(0) &= (x_0, \lambda_{x,0}) \in \mathbb{R}^{n+1} \\ \dot{x}(t) &= \begin{pmatrix} J(x(t))^+ d \\ 1 \end{pmatrix}, \quad t > 0. \end{aligned} \tag{2.27}$$

It is not possible, nevertheless, to apply PC methods yet since the parameters λ_x and λ_y parametrize different curves. However, as shown in [22] it is reasonable to match the two parameters: let x_0 be given and $\lambda_{x,0} = 0$, and let $(x_1, \lambda_{x,1})$ be an Euler step of (2.27) with a small step size $\triangle\lambda_x$. i.e., $x_1 = x_0 + \triangle\lambda_x \nu_+(x_0)$ and $\lambda_{x,1} = \triangle\lambda_x$. By construction of $\nu_+(x_0)$ and since $\triangle\lambda_x$ is small we have

$$d_i \approx \frac{f_i(x_1) - f_i(x_0)}{\triangle\lambda_x ||\nu_+(x_0)||}, \quad i = 1 \dots k. \tag{2.28}$$

This implies that $F(x_1) - F(x_0) \approx \triangle\lambda_x d||\nu_+(x_0)||$, and hence, $H(x_1, \lambda_{x,1}||\nu_+(x_0)||) \approx 0$. This way, Eqs. (2.26) and (2.27) can now be used to perform classical PC corrector methods (e.g., Euler's method [36]) to trace the solution curve.

To define a stopping criterion, authors utilize the fact that $\text{rank}(J(x_0)) = k$ (by assumption) and $\text{rank}(J(x^*)) < k$ (by definition of the critical point x^*). Nevertheless, the rank of a matrix can not be used to detect the critical point numerically. The condition number κ_2 of $J(x_0)$ can be used instead: one can for example compute

$$\kappa_2(J(x)) = ||J(x)||||J(x)^+|| = \sigma_1/\sigma_k, \tag{2.29}$$

where σ_1 and σ_k are the largest and the smallest singular values of $J(x)$, respectively, and stop the process if, for a given (large) threshold $tol \in \mathbb{R}_+$, $\kappa_2(J_i) \geq tol$. This can be done since, by the above discussion, $\kappa_2(J(x(t))) \to \infty$ when $x(t) \to x^*$.

This way of computing Pareto points shows one potential drawback of the approach, namely that the determination of the search direction by solving Eq. (2.22) gets inaccurate for points near the Pareto set due to the high condition number of $J(x_0)$ and that establishing a stopping criterion is dependent upon the condition number of $J(x)$. A solution to this problem will be addressed in the development of the new method (EDS) and shall be discussed within the next chapter.

2.3.2 Directed Search Predictor-Corrector Method

Once the main ideas of DS are established, a PC method can be developed as described in the following way.

Predictor Direction

Assume a local Pareto point $x_0 \in \mathbb{R}^n$ with $\text{rank}(J(x)) = k - 1$, and its associated weight vector α. Then by Theorem 1, α is orthogonal to the linearized Pareto front at $F(x_0)$ [27] and hence any direction orthogonal to α could be a promising predictor direction. To compute such a direction a QR factorization on α can be performed:

$$\alpha = QR = (q_1, \ldots, q_k)(r_{11}, 0, \ldots, 0)^T, \tag{2.30}$$

where $Q \in \mathbb{R}^{k\times k}$ is an orthogonal matrix and $R \in R^{k\times 1}$ with $r_{11} \in \mathbb{R}\backslash\{0\}$ is an upper triangular matrix. Since by Eq. (2.30) $\alpha = r_{11}q_1$, it follows that a well spread set of directions can be taken from any of the normalized search directions ν_i such that:

$$J(x_0)\nu_i = q_{i+1}, \quad i = 1, \ldots, k-1. \tag{2.31}$$

To orientate the curve (i.e., to determine the signum of p) the change in one of the objective values can be used. For this, the signum of the according entry of the the direction vector q_2 can be taken. If, for instance, an improvement according to f_2 is sought, then

$$p = x_0 - \text{sgn}(q_{22})\frac{t\nu_2}{||v_2||}, \tag{2.32}$$

where $t \in \mathbb{R}^+$ is the chosen step size.

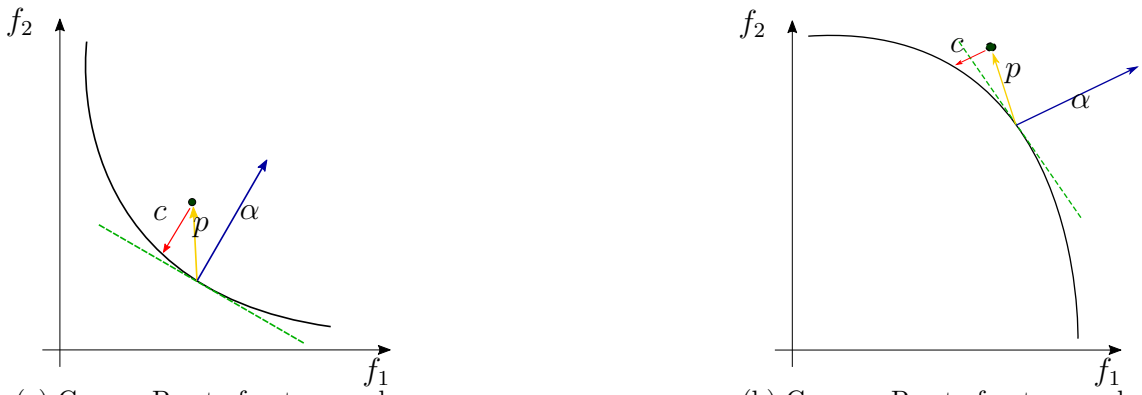
Corrector Direction

Given a predictor p, the subsequent solution along the curve can be computed by solving Eq. (2.23) numerically using p as initial value and choosing $d = -\alpha_0$, i.e., the negative of the weight from the previous solution x_0, leading to a new solution x_1.

The new associated weight vector α_1 can be updated as follows:

$$\alpha_1 = \underset{\lambda \in \mathbb{R}^k}{\operatorname{argmin}} \left\{ \left\| \sum_{i=1}^{k} \lambda_i \nabla f_i(x) \right\|^2 \text{ s. t. } \lambda_i \geq 0, i = 1, \ldots k, \sum_{i=1}^{k} \lambda_i = 1 \right\}. \tag{2.33}$$

Figure 2.6 displays a single iteration of the DS method, p stands for the predictor direction, while c stands for the corrector direction. It can be seen from the images that even though a movement along a linearization of the Pareto front at $f(x_0)$ is desired, it is not always possible to move in such direction and hence predictors usually end up above the Pareto front, making the use of corrector steps necessary in most cases.



(a) Convex Pareto front example

(b) Concave Pareto front example

Figure 2.6: Examples of the Directed Search predictor-corrector method

2.4 Direct Zig Zag Method

The Direct Zig-Zag (DZZ) method [41] is a continuation method for solving discrete or mixed-integer bi-objective problems. The DZZ method searches Pareto optimal solutions along a zig-zag path close to the Pareto front. The local zig-zag path is identified based on a pattern search idea (e.g. Hooke-Jeeves method [42]) in which the search procedure only compares function values without computing the gradients of the objective functions. Thus, the DZZ method can, in general, be applied for black-box discrete or mixed-integer MOPs where the objective functions can be evaluated through numerical or simulation processes. The DZZ method guarantees local Pareto optimality of the solutions due to the neighborhood search inside the pattern search procedure. The method consists of two parts.

In the first part, a First Pareto Solution (FPS) is computed, from a starting point x_0, by means of a modified version of the well known pattern search method [42]. This FPS is the Pareto point which minimizes f_1 while attaining the smallest value of f_2.

The second part is performed in an iterative way. For each Pareto solution x_0^* the method looks for a neighboring solution x_1 that increases the value of f_1, i.e., $f_1(x_1) > f_1(x_0^*)$, this is called a *zig step*. Since it is assumed that the continuation will start at the Pareto point that minimizes f_1 (FPS) it is logical to think that the only direction to keep moving is the one that increases the values of f_1. From x_1 a pattern search like strategy is applied in order to find a non-dominated solution x_1^*, such that $x_1^* \neq x_0^*$, this is a *zag step*. Figure 2.7 displays a simple example of the application of the DZZ method. The first phase, namely the FPS phase goes from $f_(x_0)$ to $f(x_1)$. From $f(x_1)$ onwards, zig and zag steps are iteratively applied.

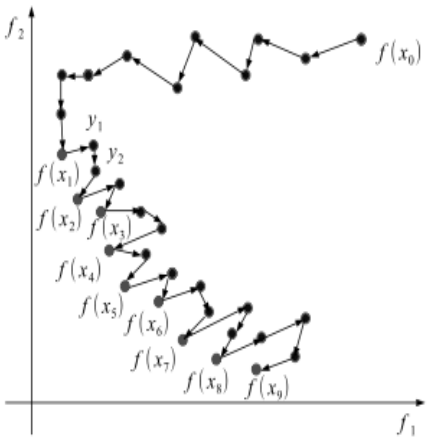


Figure 2.7: DZZ method example

It is remarkable that the method does not use gradient information making it very efficient in terms of function evaluations. Nevertheless, it is only able to solve BOPs so far, making this its biggest disadvantage.

2.5 Multi-objective Optimization Evolutionary Algorithms

Multi-objective Optimization Evolutionary Algorithms (MOEAs) use a population of solutions (called individuals) which are "evolved" through special operators inspired by the evolution of species [43, 44, 37]. This evolution process is expected to lead the

individuals towards the Pareto front. This evolutionary search process is influenced by the following main components of a MOEA:

- An *encoding* of solutions to the problems as a chromosome.
- A *function* to evaluate the fitness, or survival strenght of individuals.
- A *population* of individuals (solutions).
- A *generation counter*.
- *Initialization* of the initial population.
- *Selection* operators.
- *Crossover* operators.

Algorithm 1 shows how these components are combined to form a generic MOEA. The steps of a MOEA are applied iteratively until some stopping condition is satisfied. Each iteration is referred to as a generation.

Algorithm 1 Generic MOEA

Input: t, n_x, n_s
Output: A set $\mathbf{P}$ of Pareto points.

1: Let $t = 0$ be the generation counter
2: Create and initialize an n_x-dimensional population, $\mathbf{P}(0)$, to consist of n_s individuals
3: **while** stopping conditions not true **do**
4: Evaluate the fitness, $F(x_i(t))$, of each individual, $x_i(t)$
5: Perform crossover to create offspring
6: Select the new population, $\mathbf{P}(t+1)$
7: Advance to the new generation, i.e. $t = t + 1$
8: **end while**

MOEAs are widely used because they provide global convergence and are relatively easy to implement. Among the most widely known MOEAs are NSGA-II [45] and MOEA-D [46]. One drawback of all MOEAs however is their relatively slow convergence rates, making them impractical for some real world MOPs, where function evaluations may entail intense computing. As for MMOPs, MOEAs tend to round off the values of the solutions closest to the feasible points for Eq. (2.5) which can lead to suboptimal solutions [41].

2.6 Performance Indicators

As stated before, the solution set of a MOP (the Pareto set) is not given by a single point but typically forms a $(k-1)$-dimensional manifold where k is the number of objectives involved in the MOP. Hence, a natural question that arises is how to measure the performance of an algorithm aiming for the approximation of the entire Pareto set and its image, the Pareto front. In practice, it is always desirable to to attain an approximation that (i) covers the entire Pareto front uniformly (which accounts for spread) and (ii) lies exactly over the Pareto front (which accounts for convergence).

A performance indicator is a mapping that associates a scalar value to the Pareto set/front computed by some algorithm. Such scalar value *indicates* the quality of the solution (Pareto set/front). Hence, by using performance indicatorsthe quality of the solutions computed by different algorithms can be compared. Examples of performance indicator are the the Generational Distance (GD) [47], the Inverted Generational Distance (IGD) [48] and the Averaged Hausdorff Distance ($\triangle_p$) [33]. We will now briefly describe the $\triangle_p$ performance indicator.

Definition 2.6.1 (Archive) *An archive $\mathcal{A}$ of size l contains l elements in $\mathbb{R}^n$ which are mutually non-dominated, i.e.*

$$\begin{aligned} \mathcal{A} = a_1, \ldots, a_l, \quad a_i \in \mathbb{R}^n, \quad i = 1, \ldots, l \\ : a_i \not\prec a_j \forall i = 1, \ldots, l : \forall j = 1, \ldots, l, \quad j \neq i. \end{aligned} \tag{2.34}$$

Definition 2.6.2 (Distance from a point to a set) *The distance between a point $b \in \mathbb{R}^n$ and a set $\mathcal{A} \subset \mathbb{R}^n$ is defined as:*

$$dist(b, \mathcal{A}) := \inf_{a \in \mathcal{A}} ||b - a||_p, \tag{2.35}$$

where $|| \cdot ||_p$ is the p norm.

Definition 2.6.3 (Semi-distance between two sets) *The semi-distance between two sets $\mathcal{A} \subset \mathbb{R}^n$ and $\mathcal{B} \subset \mathbb{R}^n$ is defined as:*

$$dist(\mathcal{B}, \mathcal{A}) := \sup_{b \in \mathcal{B}} dist(b, \mathcal{A}). \tag{2.36}$$

Note that dist$(\mathcal{A}, \mathcal{B})$ is not symmetric, i.e., it does not necessary hold $dist(\mathcal{B}, \mathcal{A}) = dist(\mathcal{A}, \mathcal{B})$ for all sets.

2.6.1 Generational Distance

The Generational Distance (GD) indicator measures the averaged distance from image of the archive towards the Pareto front. Given two finite sets $\mathcal{A} = a_1, \ldots, a_N$ and $\mathcal{B} = b_1, \ldots, b_M$, and using *dist* from Definition 2.6.2, the GD indicator as proposed in [47] can be computed as follows:

$$GD(\mathcal{A}, \mathcal{B}) := \frac{1}{N} \left(\sum_{i=1}^{N} \text{dist}(a_i, B)^p \right)^{\frac{1}{p}}. \tag{2.37}$$

The GD indicator takes the distance from each member of the set $\mathcal{A}$ to the closest point in the set $\mathcal{B}$. Assume set $\mathcal{A}$ is the outcome of a given multi-objective optimization algorithm for a certain MOP F and that $\mathcal{B}$ is discretized version of the real Pareto front of F. Hence, GD measures *convergence* to the Pareto front (a GD value of zero means all solutions of $\mathcal{A}$ lie on the Pareto front).

However, GD is not well suited for measuring the *spread* of the solutions. For example, an archive with a single Pareto optimal solution would get the best value of the GD indicator.

2.6.2 Inverted Generational Distance

Proposed in [48] by Cruz and Coello. Inverted Generational Distance (IGD) can be seen as a complimentary indicator of GD. It is defined as follows:

$$IGD(\mathcal{A}, \mathcal{B}) := \frac{1}{M} \left(\sum_{i=1}^{M} \text{dist}(b_i, \mathcal{A})^p \right)^{\frac{1}{p}}, \tag{2.38}$$

where $\mathcal{A} = a_1, \ldots, a_N$ and $\mathcal{B} = b_1, \ldots, b_M$ are two finite sets.

Assuming again that the set $\mathcal{A}$ is the outcome of a given multi-objective optimization algorithm for a certain MOP F, and that $\mathcal{B}$ is discretized version of the real Pareto front of F. The IGD value denotes how "well" the archive $\mathcal{A}$ covers the given discretization of the Pareto front $\mathcal{B}$ (spread of solutions). The following scenario, nevertheless, poses a trouble for the IGD indicator: let the archive $\mathcal{A}$ contain more points than the discretization of the Pareto front $\mathcal{B}$, also let that for each point $a_i \in \mathcal{A}$ there is a point $b_j \in \mathcal{B}$ such that $\text{dist}(a_i, b_j) = 0$. Hence, all the remaining points in $\mathcal{A}$ are not taken into account and can lie wherever in the objective space.

Finally, it can be seen from the definitions of GD and IGD that

$$GD(\mathcal{A}, \mathcal{B}) = IGD(\mathcal{B}, \mathcal{A}) \tag{2.39}$$

for all finite sets $\mathcal{A}$ and $\mathcal{B}$.

2.6.3 Hausdorff Distance

Let $\mathcal{A}$, $\mathcal{B} \in \mathbb{R}^n$, then, using dist as in Definition 2.6.3, the Hausdorff distance [49] between $\mathcal{A}$ and $\mathcal{B}$ is defined as:

$$d_H(\mathcal{A}, \mathcal{B}) := \max(\text{dist}(\mathcal{A}, \mathcal{B}), \text{dist}(\mathcal{B}, \mathcal{A})). \tag{2.40}$$

The Hausdorff distance has a couple of interesting advantages over some other performance indicators. For instance, d_H defines a metric in the mathematical sense and as such the *triangle inequality*, which says that given the sets $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$, the distance from $\mathcal{A}$ to $\mathcal{C}$ via $\mathcal{B}$ is at least as great as from $\mathcal{A}$ to $\mathcal{C}$ directly, is satisfied. Another major advantage of d_H is that, assuming $\mathcal{A}$ is the outcome of a given multi-objective optimization algorithm for a certain MOP F and that $\mathcal{B}$ is discretized version of the real Pareto front of F, then a low value $d_H(\mathcal{A}, \mathcal{B})$ of the distance between the outcome set $\mathcal{A}$ and the Pareto front $\mathcal{B}$ gives a clear idea of the approximation quality of $\mathcal{A}$. On the other hand, the Hausdorff distance is yet scarcely as an indicator, the apparent reason is that d_H penalizes the largest outlier of the candidate set, which makes "good" approximations that contain at least one outlier to appear "bad". Hence, a large value of $d_H(\mathcal{A}, \mathcal{B})$ can indicate either that $\mathcal{A}$ is indeed a bad approximation of $\mathcal{B}$ or that $\mathcal{A}$ is a good approximation containing some outliers.

2.6.4 Averaged Hausdorff Distance $\triangle_p$

The $\triangle_p$ performance indicator [33] can be seen as an averaged version of the Hausdorff distance. It measures, by combining slightly modified versions of the GD and the IGD indicators, the distance of the outcome set of the algorithm $\mathcal{A}$ to the set of interest $\mathcal{B}$.

Definition 2.6.4 (Averaged Hausdorff Distance $\triangle_p$)

$$GD_p(\mathcal{A}, \mathcal{B}) := \left(\frac{1}{N} \sum_{i=1}^{N} dist(a_i, B)^p \right)^{\frac{1}{p}}$$

$$IGD_p(\mathcal{A}, \mathcal{B}) := \left(\frac{1}{M} \sum_{i=1}^{M} dist(b_i, \mathcal{A})^p \right)^{\frac{1}{p}} \tag{2.41}$$

$$\triangle_p := \max(GD_p(\mathcal{A}, \mathcal{B}), IGD_p(\mathcal{A}, \mathcal{B})).$$

It is demonstrated in [33] that $\triangle_p$ is a semi-metric for $1 \leq p < \infty$ and a metric for $p = \infty$. Furthermore, for $p = \infty$ the $\triangle_p$ coincides with the Hausdorff distance d_H. For $p < \infty$ the distances between sets are averaged, therefore, the choice of the p-norm is key to handle the outlier trade off: the smaller p, the higher the averaging effect and the lower the influence of single outliers. If, on the other hand, p is increased, outliers stronger influence the value of $\triangle_p$.

Chapter 3

The Enhanced Directed Search Method

In this chapter we will describe in detail the Enhanced Directed Search (EDS) method for the treatment of MMOPs. The EDS method is a PC algorithm based on the previously described DS method [22]. Its main task is to follow a fixed direction in objective space. For this purpose, an alternative version of the map (2.22), in page 15, is used. This alternative not only preserves the steering properties of DS but also provides a more reliable way of detecting critical points than condition (2.29) in page 17. Following a direction in objective space allows us to implement a PC method capable of handling MOPs and MMOPs. The resulting method can be applied to problems with any number of objectives using only first order gradient information. Another important feature of the EDS method is the implementation of a mechanism that allows it to use neighborhood information in order to save function evaluations, making EDS more efficient than the classical DS.

This chapter is organized as follows: Section 3.1 defines the mapping of a direction in objective space to a direction in parameter space. Sections 3.2 and 3.3 address predictor and corrector steps of the algorithm respectively. The strategy used for handling MOPs with more than two objectives is discussed in Section 3.4.1. The use of neighborhood information within the computation of predictors and correctors is explained in Section 3.5. Finally, modifications to the predictor and the corrector in order to handle MMOPs are reviewed in detail in Section 3.6.

3.1 Following a Direction in Objective Space

Our main task in the development of the EDS continuation method is to "follow" a certain direction in objective space. For this, we will use the work developed in

[50], which allows to map a direction from objective space $\nu \in \mathbb{R}^n$ to direction in parameter space $d \in \mathbb{R}^k$. This is achieved by solving the following single-objective optimization problem [50]:

$$\begin{aligned} \min_{\nu,\delta} \quad & \frac{1}{2}\|\nu\|_2^2 - \delta \\ \text{s. t.} \quad & J(x)\nu = \delta d, \end{aligned} \tag{3.1}$$

where $J(x) \in \mathbb{R}^{k\times n}$ is the Jacobian of F as defined in Eq. 2.1 in page 6 at the point x and $\delta \in \mathbb{R}$ is a scalar value. Let (ν^*, δ^*) be a solution of Problem (3.1) where $d \neq 0$. Then for v^* and δ^* the following propositions hold [50]:

Proposition 3.1.1

a) $v^* \neq 0 \iff \delta^* > 0 \iff \exists v \in \mathbb{R}^n : J(x)v = d$. *In that case* $v^* = \delta^* J^+(x)d$.

b) $v^* = 0 \iff \delta^* = 0$. *In that case* x *is a critical point (see Definition 2.3.1 in page 16.).*

c) $v(x)$ *and* $\delta(x)$ *are continuous mappings.*

By Proposition 3.1.1 it follows that:

- δ^* can be used as a criterion for determining critical points.
- Problem (3.1) can be used for mapping a direction in objective space to a direction in parameter space.

Problem 3.1 along with the above propositions open the possibility for steering the search in a desired direction in objective space and mapping it to parameter space. As it can be seen, this is exactly the main idea underlying DS, nevertheless Proposition 3.1.1 provides a more stable criterion for determining critical points that the one provided in [22]. Hence, by solving Problem (3.1) for a point $x_0 \in \mathbb{R}^n$ and a direction in objective space $d \in \mathbb{R}^k$, one obtains a direction in parameter space $\nu \in \mathbb{R}^n$ such that $F(x_i)$, where $x_i = x_0 + t\nu$, moves along d for a suitable step size t, and a scalar $\delta \in \mathbb{R}$, where $\delta = 0$ when x_0 is a critical point. The aforementioned ideas serve as the basis for the predictor and corrector steps in the EDS method. Here we would like to stress that for this particular application we have used an interior-point algorithm (as implemented by the fmincon function in Matlab [51]) for the solution of problem (3.1).

3.2 Predictor

In this section we will analyze the steps for the computation of the predictor direction. Assume we are given a (local) Pareto point and its associated convex weight $\alpha \in \mathbb{R}^k$, further, we assume that $\text{rank}(J(x)) = k - 1$. Then it is known by Theorem 2 in page 9 that α is orthogonal to the linearized Pareto front at $F(x)$. Thus, a search orthogonal to α could be promising to obtain new predictor points. To obtain such directions a QR-factorization of α can be computed, i.e.,

$$\alpha = QR, \tag{3.2}$$

where $Q = (q_1, \ldots, q_k) \in \mathbb{R}^{k\times k}$ is an orthogonal matrix and $R = (r_{11}, 0, \ldots, 0)^T \in R^{k\times 1}$ with $r_{11} \in \mathbb{R}\backslash\{0\}$. Since by Eq. (3.2) $\alpha = r_{11}q_1$, it follows that a well spread set of directions can be taken from any of the normalized search directions ν_i such that:

$$J(x)\nu_i = q_{i+1}, \quad i = 1, \ldots, k-1. \tag{3.3}$$

It follows by the rank of $J(x)$ that the vectors $q_2, \ldots, q_k$, are in the image of $J(x)$ and hence Eq. (3.1) can be solved for each q_i where $i \in \{2, \ldots, k\}$. Please note that such predictor directions $\nu_i = \delta_i J^+(x)q_i$ do not have to be tangent to the *Pareto set*. Instead, $\nu = \delta J(x)^+ q$ points along the linearized Pareto front (see Figure 2.2). Since by Proposition 3.1.1 ν is the most greedy solution of Problem (3.1) with respect to q, we can expect that the image $F(p)$, where $p \in \mathbb{R}^n$ is the chosen predictor, is also close to the Pareto front, and thus, that only few iteration steps are required to correct back to the front. Hence, a set of predictors can be computed in the following way:

$$\begin{aligned} p_{i+} &= x + \hat{t}\frac{\nu_i}{||\nu_i||} \\ p_{i-} &= x - \hat{t}\frac{\nu_i}{||\nu_i||}, \end{aligned} \tag{3.4}$$

where $\hat{t}$ is the chosen step size. Note that both, the positive and the negative predictors have to be considered. This is done to ensure that the search is done in every possible direction and hence that a full covering of the Pareto front is obtained at the end of EDS execution.

Finally, to obtain the predictor p at the current point x, we need to select a suitable step size. In this study we are particularly interested in an evenly distributed set of solutions along the Pareto front. That is, at least for two consecutive solutions x_{i+1}

and x_i we want that

$$||F(x_{i+1}) - F(x_i)|| \approx \tau, \tag{3.5}$$

where $\tau > 0$ is a user specified value. For this, we follow the suggestion made in [27] and take the step size

$$\hat{t} = \frac{\tau}{||J(x)\nu||}. \tag{3.6}$$

The entire predictor phase is shown in Algorithm 1

Algorithm 1 Obtain predictors

Input: Initial point $x \in \mathbb{R}^n$, predictor direction in objective space $d \in \mathbb{R}^k$
Output: A set of predictors p_+ and p_-

1: Compute $J(x)$
2: Compute direction ν_p by solving Problem (3.1)
3: Compute predictor step size $\hat{t}$ by Eq. (3.6)
4: Set $p_+ = x + \hat{t}\nu$ and $p_- = x - \hat{t}\nu$
5: return $[p_+, p_-]$

Figure 3.1 exemplifies the predictor phase. First, for the local Pareto point x_i the α vector is computed, then a direction d_+ orthogonal to α is computed. Finally, after executing Algorithm 1, two predictors p_+ and p_- are obtained.

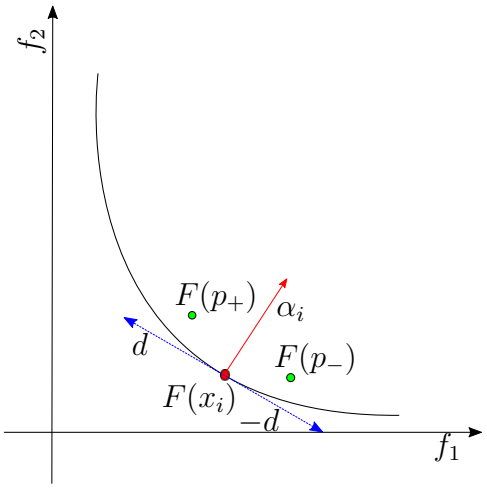


Figure 3.1: Computation of the predictors

3.3 Corrector

The goal of the corrector phase is to ensure that the resulting solution is on the efficient set (i.e., it is at least a local Pareto point). Since typically a predictor can not even guarantee local optimality, some corrector steps may be necessary to ensure that a solution near to the Pareto front is computed at the end of the iteration. Therefore, our new target is to bring (or correct) the predicted point back to the solution manifold, which is certainly *not* an easy task. Nevertheless, due to the smoothness assumptions considered in this study; namely that a good property of predictors is that they remain close to the Pareto set provided that the step size t used by the predictor is not *too large*, it is expected that few corrector steps are needed.

For the computation of the corrector, a new directed search descent method was developed. Assume we are given a predictor p along with the weight vector α associated to the initial point x from which p was computed. Furthermore, since by assumption $F(p)$ is *close* to $F(x)$ it makes sense to use the opposite direction of α, i.e., $-\alpha$, in order to move back to the Pareto front (see Figure 3.2).

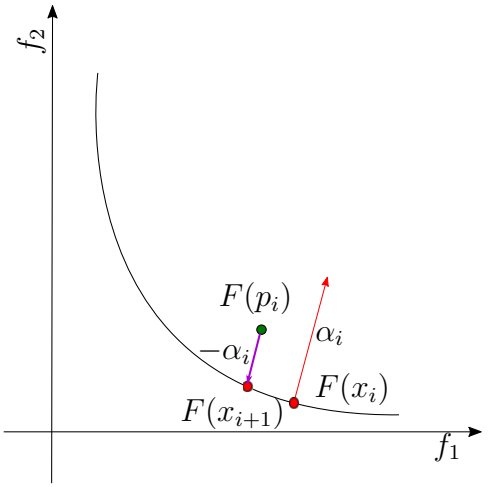


Figure 3.2: $-\alpha_i$ taken as steering direction for the corrector

Thus, given $p \in \mathbb{R}^n$ and $-\alpha \in \mathbb{R}^k$, a direction $\nu_c \in \mathbb{R}^n$ that moves along $-\alpha$ can be computed by solving Eq. (3.1) for p and $-\alpha$. A movement along ν_c (and hence in $-\alpha$ direction in objective space) is then performed in the following way:

$$c_i = p_i + t\frac{\nu_c}{||\nu_c||}, \tag{3.7}$$

where t is the corrector step size. In this phase several corrector iterations may be computed until a certain stopping criterion (see Section 3.3.2) is met.

Two important issues arise now: first, how to compute a suitable step size for the corrector? And second, how to determine whenever the corrector is indeed a critical point? Both issues will be addressed in the following sections.

3.3.1 Corrector Step Size

As stated in Section 2.3 a movement along $d \in \mathbb{R}^k$ (in objective space) using the map (2.22), which maps to a movement ν in parameter space

$$x_{i+1} = x_i + t\nu, \tag{3.8}$$

is only possible for *sufficiently small* values of t. Let $\tilde{d} = F(x_{i+1}) - F(x_i) \in \mathbb{R}^k$, be the vector in objective space obtained after moving along ν direction for a given step size t, also let

$$\beta = \cos^{-1}\left(\frac{\langle \tilde{d}, d \rangle}{||\tilde{d}||||d||}\right), \tag{3.9}$$

be the angle between $\tilde{d}$ and d. It was observed during our experiments that as t increases the deviation between $\tilde{d}$ and d, measured by the angle β, also increases. Figure 3.3 illustrates this behavior. For increasing step sizes t, a deviation in the current direction $\tilde{d}$ can be observed with respect to the original direction d.

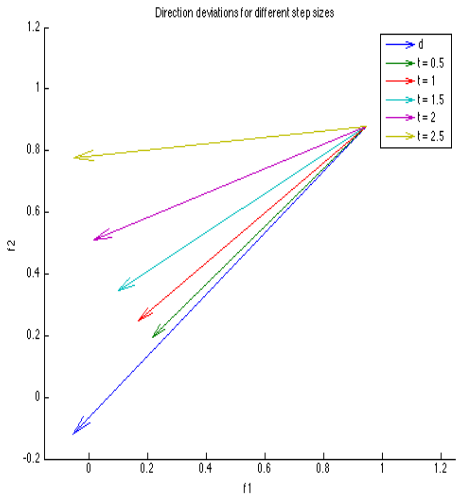


Figure 3.3: Deviation of the direction $\tilde{d}$, obtained by following ν, with respect to d.

Therefore, the step size for the corrector plays an important role in the overall performance of the method.

Assume we are given a predictor p along with a direction in parameter space ν and a direction d in objective space. Let $\tilde{d} = F(c) - F(p)$ where $c = p + t\nu$. A suitable step size t provides the largest progress in direction $\tilde{d}$ while keeping angle β bellow a user defined threshold. For our convenience we will define $\epsilon \in [0, 1]$ as the aforementioned threshold where $\epsilon = cos(\beta)$, therefore keeping the angle itself bellow a threshold $cos^{-1}(\epsilon)$. Furthermore we would like that the corrector c at least weekly dominates the previous point p. Hence, the optimal step size t can be computed by solving the following constrained single objective optmization problem (SOP):

$$\begin{aligned} \underset{t\in\mathbb{R}}{\text{minimize}} \quad & -||\tilde{d}|| \\ \text{s. t.} \quad & \\ & min(\tilde{d}) \leq 0 \\ & \epsilon - \frac{\langle \tilde{d}, d\rangle}{||\tilde{d}||||d||} \leq 0, \end{aligned} \tag{3.10}$$

where $\tilde{d} = F(p + t\nu) - F(p)$. The objective function of Problem (3.10) will try to maximize the movement along $\tilde{d}$. The first constraint ensures that the newly computed point $F(c)$ at least weakly dominates the previous one $F(p)$. We will now bring our attention to the second constraint.

Assume that a certain deviation from direction d, measured by $\epsilon \in [0, 1]$, is permitted. Then any direction $\tilde{d} = F(c) - F(p)$ that fulfills the constraint

$$\epsilon - \frac{\langle \tilde{d}, d\rangle}{||\tilde{d}||||d||} \leq 0 \tag{3.11}$$

points in direction d with a maximum deviation of $\beta = cos^{-1}(\epsilon)$ degrees. Thus, the smaller the value of ϵ, the greater the permitted angle between $\tilde{d}$ and d (and thus the deviation between them) is. Also note that by construction of constraint (3.11) angles $\beta \in (\frac{\pi}{2}, \frac{3\pi}{2})$, and thus solutions that are dominated by $F(p)$, are automatically discarded. Figure 3.4 depicts constraint (3.11), any point in between the dashed arrows is an acceptable corrector point c, recall that $\beta = cos^{-1}(\epsilon)$.

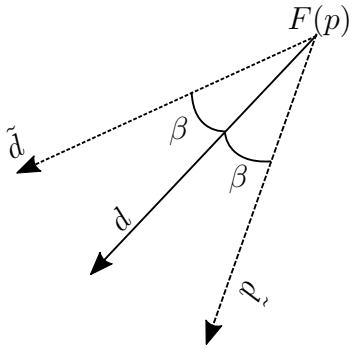


Figure 3.4: Acceptable deviations for the corrector c

Therefore, by solving Problem (3.10) a suitable step size for the corrector can be obtained. Nevertheless, solving Problem (3.10) would directly impact on the efficiency of EDS method. Instead, a more efficient backtracking-like strategy has been adopted.

Assume we are given ϵ and an initial step size t_0 (usually the last suitable step size computed). We test whether t_0 fulfills the constraints of Problem (3.10), if it does we take t_0 as step size for the corrector, otherwise we shrink t_0 and try again until a suitable step size t_0 is found or until t_0 is bellow a user defined threshold. In case t_0 is suitable at the first try we set $t_1 = 2t_0$ and use it as initial guess for the next corrector step.

Finally, it is worth to mention that although the value of ϵ is a user defined and problem dependent parameter, experimental results have shown that $\epsilon \in [0.6, 0.9]$ improve the overall performance of the EDS method. Algorithm 2 shows how to compute a suitable step size for the corrector given a direction ν in parameter space, a direction d in objective space, and an initial point x_0.

Algorithm 2 Corrector step size

Input: Current point $x \in \mathbb{R}^n$ along with its function value $F(x)$, step size guess $t \in \mathbb{R}^+$, $\nu \in \mathbb{R}^n$ direction and minimum step size threshold min_t.
Output: A step size t, a new point $x = x + t\nu$ along with its function value $F(x)$, and a flag b_first indicating whether step size h was suitable at first try.

```
Compute a random shrinking factor ρ ∈ [0.1, 0.6]
b_cont = 1
b_first = 1
x_init = x
while b_cont ≠ 0 do
    x = x_init + tν
    Evaluate F(x)
    if F(x) satisfies the constraints of Problem (3.10) then
        b_cont = 0
    else
        Shrink t by ρ factor, i.e., t = ρt
        b_first = 0
    end if
    if t < min_t then
        t = 0
        b_cont = 0
    end if
end while
return [x, F(x), t, b_first]
```

3.3.2 Determining Critical Points

Now that we have a promising search direction $\nu \in \mathbb{R}^n$ and a suitable step size $t \in \mathbb{R}$ the remaining task of the corrector step is to determine whether a corrector point c is a critical point or not. We are interested in the detection of such points since they are strong candidates of being Pareto points (see Definition 2.3.1 in page 16). Recall, by Proposition 3.1.1, that $\delta = 0$ when c is a critical point and that $\delta(x)$ is a continuous map. Therefore, the value of δ can be used for determining if a corrector c is indeed a critical point. Two thresholds for δ can be defined: max_$\delta > 0$ and $0 <$ min_$\delta <$ max_δ. The use of both thresholds and the entire corrector phase is described in Algorithm 3.

Algorithm 3 Compute corrector

Input: Direction $-\alpha \in \mathbb{R}^k$, predictor point $p \in \mathbb{R}^n$, thresholds min_$\delta \in \mathbb{R}^+$ and max_$\delta \in \mathbb{R}^+$

Output: A point $x \in \mathbb{R}^n$ along with its function value $F(x)$ and a flag $b \in \{0, 1\}$ indicating whether x is a critical point or not.

$x = p$
while true **do**
    Compute $J(x) \in \mathbb{R}^{k \times n}$
    Compute ν, s.t. $J\nu = d$, by solving Problem (3.1)
    if $\delta <$ min_δ **then**
        return $[x, F(x), b = 1]$
    end if
    Obtain $[x, F(x), h, b\_first]$ by Algorithm 2
    if $h = 0$, i.e. no suitable step size could be computed **then**
        if $\delta <$ max_δ **then**
            return $[x, F(x), b = 1]$
        else
            return $[x, F(x), b = 0]$
        end if
    else
        if $b\_first = 1$ **then** $t = 2t$
        end if
    end if
end while
return $[x, F(x), b = 0]$

As it can be seen the thresholds min_δ and max_δ are critical for the robustness and performance of EDS method. Further investigation on this topic is left for future research.

Finally after a new critical point is computed, its associated weight vector α can be updated as follows ([52]):

$$\alpha = \arg\min_{\lambda \in \mathbb{R}^k} \left\{ \left\| \sum_{i=1}^{k} \lambda_i \nabla f_i(x) \right\|^2 \text{ s. t. } \lambda_i \geq 0, i = 1, \ldots k, \sum_{i=1}^{k} \lambda_i = 1 \right\}. \tag{3.12}$$

3.4 The Enhanced Directed Search Method

We are now in the position to put predictor and corrector methods together into a continuation algorithm that we call Enhanced Directed Search (EDS). The EDS method is able to "follow" the curve (or manifold) of Pareto points of a MOP. This procedure perfectly fits into the category of PC methods. There are, nevertheless, still two issues that need to be solved before defining the algorithm of EDS: first, how to identify the parts of the manifold that have already been covered by EDS? And second, when and how should the glseds method be stopped? Answers to these questions are given in the following sections.

3.4.1 Determining Points Already Covered by EDS

The issue of determining points already computed by the EDS method arises already for BOPs. Recall that the predictor phase computes two predictors: one that maps to direction $d \in \mathbb{R}^k$ and another that maps to direction $-d \in \mathbb{R}^k$. This is done to ensure that by the end of EDS execution a well spread discretization of the Pareto front is obtained.

Let $x \in \mathbb{R}^n$ be a local Pareto point. Now, assume that a set of predictors $p_+ = x + \hat{t}\nu$ and $p_- = x - \hat{t}\nu$ are computed for a fixed value of $\hat{t}$ and that from each of the predictors a new critical point can be reached by means of a corrector step. Let x_1 be one of such new critical points, shall we compute again a set of predictors and correctors with the same $\hat{t}$ from x_1 it is most likely that one of the new critical points, say x_2, will be within a "small" neighborhood of x and hence if the process is repeated with x_2 it is clear to see that a set of "repeated" points will be computed. Therefore, a mechanism to avoid computing such "repeated" points should be developed.

To achieve this, it is crucial to provide an efficient method to keep track of the computed solutions. To this end, the method proposed in [53] is used. According to [53] the objective space can be divided into a set of *small* boxes where each box $\mathcal{B} \subset \mathbb{R}^k$ is represented by a center $\hat{c} \in \mathbb{R}^k$ and a radius $\hat{r} \in \mathbb{R}^k$. Therefore, the covering box $\mathcal{B}(F(x))$ of a given solution $F(x)$ can be unequivocally determined.

The subdivision of the objective space into boxes is done in the following way: suppose that every function in Problem (2.1) is restricted to a certain range in objective space

$$l_j^{\mathcal{F}} \leq f_j(x) \leq u_j^{\mathcal{F}}, \qquad j = 1, \ldots, k. \tag{3.13}$$

The objective space is then contained in

$$\mathcal{Q}_{\mathcal{F}} = [l^{\mathcal{F}}, u^{\mathcal{F}}] = [l_1^{\mathcal{F}}, u_1^{\mathcal{F}}] \times [l_2^{\mathcal{F}}, u_2^{\mathcal{F}}] \times \ldots \times [l_k^{\mathcal{F}}, u_k^{\mathcal{F}}] \in \mathbb{R}^{k\times 2}. \tag{3.14}$$

Hence, each box $\mathcal{B}$ can be determined by a center $\hat{c} \in \mathbb{R}^k$ and a radius $\hat{r} \in \mathbb{R}^k$:

$$\mathcal{B}(\hat{c}, \hat{r}) = \left\{ F(x) \in \mathcal{Q}_{\mathcal{F}} \middle| \begin{matrix} \hat{c}_j - \hat{r}_j \leq f_j(x) < \hat{c}_j + \hat{r}_j, & \text{if } \hat{c}_j + \hat{r}_j < u_j^{\mathcal{F}} \\ \hat{c}_j - \hat{r}_j \leq f_j(x) \leq \hat{c}_j + \hat{r}_j, & \text{if } \hat{c}_j + \hat{r}_j = u_j^{\mathcal{F}}, \end{matrix} \quad \forall j = 1, \ldots, k \right\}. \tag{3.15}$$

Each of these boxes can be subdivided with respect to the i-th coordinate leading to two boxes that completely cover the old one. That is, $\mathcal{B}^-(\hat{c}_-, \hat{r})$ and $\mathcal{B}^+(\hat{c}_+, \hat{r})$ where

$$\hat{r}_j = \begin{cases} \hat{r}_j, & i \neq j \\ \hat{r}_j/2, & i = j \end{cases} \quad \text{and} \quad \hat{c}_j^{\pm} = \begin{cases} \hat{c}_j, & i \neq j \\ \hat{c}_j \pm \hat{r_j}/2, & i = j \end{cases}. \tag{3.16}$$

Subsequently, after h subdivisions of the current set of boxes and by cyclically choosing the reference coordinate, i.e.,

$$j_i = (i-1) \bmod k + 1, \qquad i = 1, \ldots, h, \tag{3.17}$$

we finish with a partition of $\mathcal{Q}_{\mathcal{F}}$ into boxes of radius

$$\hat{r}_j = \begin{cases} \frac{u_j^{\mathcal{F}} - l_j^{\mathcal{F}}}{2^{\left\lfloor \frac{h-1}{k} \right\rfloor + 2}}, & j \leq (h-1) \bmod k + 1 \\ \frac{u_j^{\mathcal{F}} - l_j^{\mathcal{F}}}{2^{\left\lfloor \frac{h-1}{k} \right\rfloor + 1}}, & j > (h-1) \bmod k + 1 \end{cases}, \quad j = 1, \ldots, k, \tag{3.18}$$

where for every point $F(x) \in \mathcal{Q}_{\mathcal{F}}$ there exists exactly one covering box with center in

$$\hat{c}_j = \begin{cases} l_j^{\mathcal{F}} + 2\hat{r}_j \left\lfloor \frac{f_j(x) - l_j^{\mathcal{F}}}{2\hat{r}_j} \right\rfloor + \hat{r}_j, & f_j(x) \notin u_j^{\mathcal{F}} \\ l_j^{\mathcal{F}} + 2\hat{r}_j \left\lfloor \frac{f_j(x) - l_j^{\mathcal{F}}}{2\hat{r}_j} \right\rfloor - \hat{r}_j, & f_j(x) \in u_j^{\mathcal{F}} \end{cases}, \quad j = 1, \ldots, k. \tag{3.19}$$

We can thus maintain a collection $\mathcal{B}(\mathcal{F}^*)$ to store the boxes corresponding to the solution's images in $\mathcal{F}^*$. An efficient implementation of this data structure can be

achieved by considering a binary tree $\mathcal{B}(\mathcal{F}^*)$ where each level corresponds to one subdivision step and each leaf to one box. Considering $\mathcal{B}(\mathcal{Q}_\mathcal{F}, h)$ the full set of boxes obtained after h subdivisions, it is not hard to see that $\mathcal{B}(\mathcal{Q}_\mathcal{F}, h)$ is defined by a complete binary tree of height h and that any arbitrary tree $\mathcal{B}(\mathcal{F}^*) \subset \mathcal{B}(\mathcal{Q}_\mathcal{F}, h)$ is usually not complete. Moreover, using this scheme, the memory requirements seem to grow linearly in the number of objectives and solutions. On the other hand, the operation of insertion as well as verifying whether a box (leaf) belongs to the set (tree) can be implemented in $\mathcal{O}(h)$. Figure 3.5 depicts the data structure used for keeping track of the computed solutions, each leaf of the three represents a subdivision of the objective space.

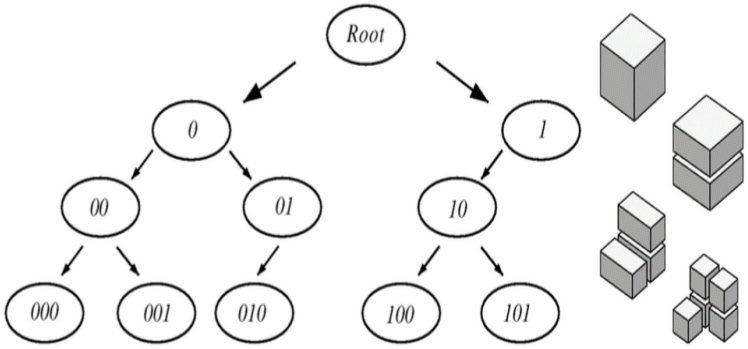


Figure 3.5: The data structure used for the representation of the solution set.

The value of h plays a key role in the performance of the method. In this work, since we are interested in obtaining a set of evenly distributed solutions along the Pareto front, where the distance between neighbors is approximately $\tau > 0$, we will follow the suggestion made in [53] and take

$$h = k \left\lceil log_2 \frac{max(u_\mathcal{F} - l_\mathcal{F})}{\tau} \right\rceil. \tag{3.20}$$

By using the aforementioned approach, the problem of detecting parts of the manifold that have already been computed reduces to determining whether a solution $F(x)$ belongs to a determined box $\mathcal{B}(F(x))$ and to determine whether that box already contains a solution. This can be achieved by Algorithm 4. The method ensures that the covering of the box of a given solution is added to the tree and returns whether the corresponding leaf was *really* inserted or already belonged to the collection. For further details of this approach and the underlying data structure please refer to [53].

Algorithm 4 BoxContains

Input: Binary tree $\mathcal{B}(\mathcal{F}^*) \subset \mathcal{B}(\mathcal{Q}_{\mathbb{F}}, h)$, $F(x) \in \mathbb{R}^k$
Output: Box $\mathcal{B}$ where $\mathcal{F}^*$ belongs to and a binary flag $b \in \{0, 1\}$ indicating whether the point $\mathcal{F}^*$ was inserted or not into the box.

$b = 0$
Starting at the root of $\mathcal{B}(\mathcal{F}^*)$
for $i = 1, \ldots, h$ **do**
    Compute $j = (i - 1) \bmod k + 1$
    Compute $c_j = l_j^{\mathcal{F}} + (u_j^{\mathcal{F}} - l_j^{\mathcal{F}})/2$
    if $f_j(x) < c_j$ **then**
        if there is no left child **then**
            Create a new left child
            $b = 1$
        end if
        Move one level down to the left child
        Set $u_j^{\mathcal{F}} = c_j$
    else
        if there is no right child **then**
            Create a new right child
            $b = 1$
        end if
        Move one level down to the right child
        Set $l_j^{\mathcal{F}} = c_j$
    end if
    Compute $\hat{r} = (u_{\mathcal{F}} - l_{\mathcal{F}})/2$ and $\hat{c} = l_{\mathcal{F}} + \hat{r}$
end for
return $[\mathcal{B}(\hat{c}, \hat{r}), b]$

It is worth to mention that an additional optimization for EDS can be achieved via bypassing predictor points when their associated boxes have previously been considered (i.e., they already contain a corrector). Considering that predictors and correctors are supposed to be close, this will avoid the computation of a corrector that may be later discarded.

3.4.2 Handling Box Constrained Problems

The strategy for handling box constrained problems in the EDS method is straightforward. This strategy is widely used and consists on the following: let $x \in \mathbb{R}^n$ be a box constrained vector, i.e., $lb \leq x \leq ub$, where $ub \in \mathbb{R}^n$ and $lb \in \mathbb{R}^n$ are the upper and lower boundaries vectors for x. If any of the components of x is bigger than its corresponding component in ub, i.e., $x_i > ub_i$, we make $x_i = ub_i$. The same applies if

$x_i < lb_i$. In short, this strategy projects the value of the components that violate the constrains x_i to the n-dimensional box formed by ub and lb.

3.4.3 Stopping Criteria for the EDS Method

Our last task is to define the stopping criteria for the EDS method, which is in fact straightforward. Assume that each newly computed critical point (x) is stored in a queue $\mathcal{M}$. The EDS method takes a point x_i from $\mathcal{M}$ at each iteration and tries to compute new predictors (and hence a correctors) from this point and add the newly computed critical points to $\mathcal{M}$. It is then obvious to stop EDS whenever the queue $\mathcal{M}$ is empty. Nevertheless, if we keep adding points to $\mathcal{M}$ the queue will never be empty. Thus, criteria for adding new points to $\mathcal{M}$ should be defined.

It is indeed simple to define which points should be added to $\mathcal{M}$. Here is a list of all the conditions that will avoid a point x from being added to $\mathcal{M}$:

1. Any point whose δ value is above the user defined thresholds min_δ and max_δ
2. Any point x, whose function value $F(x)$ lies within a box that already contains a point (see Section 3.4.1). For this purpose we use Algorithm 4.
3. Any point x, whose associated weight vector $\alpha \in \mathbb{R}^k$ does not comply with the condition: $max(\alpha) \leq 1 - tol$, where $0 < tol < 0.1$ is a user defined, problem dependent tolerance. This condition excludes points that lie on the boundary of the Pareto front and thus are likely to be dominated ones.

Now we have all the necessary elements to introduce EDS PC method. Its pseudo-code is presented in Algorithm 5.

Algorithm 5 EDS Predictor-Corrector method

Input: Starting local Pareto point $x_0 \in \mathbb{R}^n$ along with is associated weight vector $\alpha \in \mathbb{R}^k$, $\tau \in \mathbb{R}$ value defining the spreadness of the solutions, $\min\_\delta \in \mathbb{R}^+$ and $\max\_\delta \in \mathbb{R}^+$ thresholds and minimum step size threshold $\min\_h \in \mathbb{R}^+$

Output: Finite size approximation of $\mathcal{P}$ and $\mathcal{F}^*$

Set $\mathcal{P} = \mathcal{P} \cup x_0$ and $\mathcal{F} = \mathcal{F} \cup F(x_0)$
Compute $J(x)$
Enqueue($\mathcal{M}$, x_0)
while $\mathcal{M}$ not empty **do**
    $x = \text{Dequeue}(\mathcal{M})$
    Compute a set of promising directions in objective space by Eq. (3.2)
    for each $q_i \in i = 2, k$ **do**
        Set $d = q_i$
        Compute a set of predictors p_+ and p_- using Algorithm 1
        for each predictor **do**
            Compute a corrector x_1 using Algorithm 3
            if x_1 does not meet any of the criteria in Section 3.4.3 **then**
                Enqueue($\mathcal{M}$, x_1)
                $\mathcal{P} = \mathcal{P} \cup x_1$
                $\mathcal{F} = \mathcal{F} \cup F(x_1)$
            end if
        end for
    end for
end while
return $[\mathcal{P}, \mathcal{F}]$

Finally, one important aspect related to the performance of EDS method has to do with the information we keep or recompute regarding the current set of solutions. For each point in the queue $\mathcal{M}$, the corresponding function and α values are stored respectively. Furthermore, the Jacobian matrix is also stored. Storing all this information has a direct impact on the memory requirements of the method. Nevertheless, since one of our goals is to make the method efficient in terms of functions evaluations we have taken this approach to avoid recomputing information.

3.5 Using Neighborhood Information

So far, the EDS method requires Jacobian information for the computation of ν via Eq. (3.1). In this section, we present an alternative way to obtain such search directions ν without explicitly computing or approximating the Jacobian. Instead, the neighborhood information is exploited and used for the approximation of ν. This

method can be viewed as a particular finite difference method [36], however, it has the advantage that the information of points already computed can be exploited in order to approximate the Jacobian matrix and hence some function evaluations can be saved. This is particularly interesting within the context of set-based optimization strategies such as MOEAs. The approach taken here was inspired by the usage of neighborhood information in the DS method [22] and the Gradient Subspace Approximation (GSA) method [34].

The general idea behind the method is as follows: given a point x_0 that is designated for local search as well as another point x_i whose function value is known that is in the vicinity of x_0, then the given information can be used to approximate the directional derivative in direction

$$\nu_i := \frac{(x_i - x_0)}{||x_i - x_0||}, \tag{3.21}$$

without any additional cost (in terms of function evaluations). That is, it holds

$$f_{\nu_i}(x_0) = \langle \nabla f(x_0), v_i \rangle = \frac{f(x_i) - f(x_0)}{||x_i - x_0||} + \mathcal{O}(||x_i - x_0||). \tag{3.22}$$

This can be seen by considering the forward difference quotient on the line search function $f_{\nu_i}(t) = f(x_0 + t\nu_i)$.

Now assume a candidate solution $x_0 \in \mathbb{R}^n$ is designated for local search and further r search directions $\nu_i \in \mathbb{R}^n$, $i = 1, \ldots, n$, are given. Then, the matrix $\mathcal{A} := JV \in \mathbb{R}^{k \times r}$, where $V = (\nu_1, \ldots, \nu_r) \in \mathbb{R}^{n \times r}$, is as follows:

$$\mathcal{A} = JV = (\langle \nabla f_i(x), \nu_j \rangle)_{i=1,\ldots,k. \quad j=1,\ldots,n}. \tag{3.23}$$

Hence, every element m_{ij} of JV is defined by the value of the directional derivative of objective f_i in direction ν_j, an can be approximated as in Eq. (3.22). An approximation to the Jacobian matrix can thus be obtained by computing

$$J \approx \mathcal{A}(V^T V)^{-1} V. \tag{3.24}$$

Crucial for the approximation of $\mathcal{A}$ is the choice of tests point x_i. If the function values of points in a neighborhood $N(x_0)$ are already known, it seems to be wise to include them to build the matrix V (see Figure 3.6).

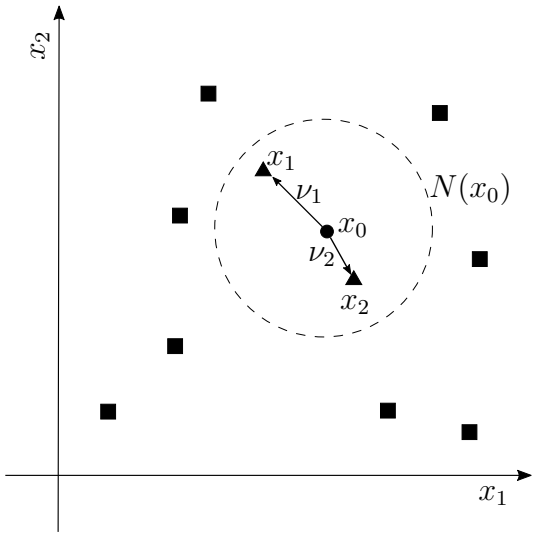


Figure 3.6: Neighbors used for the approximation of the Jacobian are spotted as triangles, squares are points outside the neighborhood $N(x_0)$

Nevertheless, it might be that further test points have to be sampled to obtain the n neighbors. More precisely, assume that we are given $x_0 \in \mathbb{R}^n$ as well as l neighboring solutions $x_1, \ldots, x_l \in N(x_0)$. One desirable property of all the remaining search directions is that they are both orthogonal to each other and orthogonal to the previous ones. In order to compute the new search directions $v_{l+1}, \ldots, v_r$, $r > 1$, one can proceed as follows: compute $V = QR = (q_1, \ldots, q_l, q_{l+1}, \ldots, q_n)R$. Then it is by construction $v_i \in span\{q_1, \ldots, q_i\}$ for $i = 1, \ldots, l$, and hence

$$\langle v_i, q_j \rangle = 0, \qquad \forall i \in \{i, \ldots, l\}, j \in \{l+1, \ldots, r\}. \tag{3.25}$$

One can thus e.g. set

$$v_{l+i} = q_{l+1}, \qquad i = 1, \ldots, r-1 \tag{3.26}$$

$$x_{l+i} = x_0 + v_{l+i}, \qquad i = 1, \ldots, r-l. \tag{3.27}$$

This way, neighborhood information can be used to approximate the value of the Jacobian. By doing this, some function evaluations can be saved by using the information computed in previous iterations. It is also important to note that the choice of r and the size of the neighborhood $N(x_0)$ has a direct impact on the quality of the method, it can be seen that for $r = k$ search directions ν_i, $i = 1, \ldots, r$, one can find a descent direction ν by solving Eq. (3.1) and using J as in Eq. (3.24) regardless of n. However, by construction it is $\nu \in span\{\nu_1, \ldots, \nu_r\}$, which means that only a

r-dimensional subspace of the $\mathbb{R}^n$ space is explored in each step. One would expect that the more search directions ν_i are taken into account, the better the choice of ν is. In fact, for the development of this work $r = n$ is used. As for the size of neighborhood, we recommend $0.01 \leq N(x_0) \leq 1$, nevertheless, it must be taken into account that this is a problem dependent parameter.

3.6 Handling Mixed-Integer Problems

The EDS method is now capable of computing the connected components of MOPs with $k \geq 2$ in an efficient way. Nevertheless, our main objective in this work is the development of a continuation method for MMOPs. A simple modification can be done to the EDS method in order to make it capable, under a certain condition on the structure of the problem, of solving MMOPs.

The aforementioned condition has to do with the search space of the problem. Recall that MMOPs are defined almost exactly as MOPs except that the parameter space is defined by a mixture of real and discrete variables (see Section 2.1.3 in page 10). The key for the mechanism that allows the EDS method to solve MMOPs is the following.

Definition 3.6.1 (Domain restriction) *Let $f : \mathbf{E} \mapsto \mathbf{F}$ be a function from a set $\mathbf{E}$ to a set $\mathbf{F}$, so that the domain of f is in $\mathbf{E}$ ($\operatorname{dom} f \subseteq \mathbf{E}$). If a set $\mathbf{A}$ is a subset of $\mathbf{E}$, then the restriction of f to $\mathbf{A}$ is the function*

$$f|_{\mathbf{A}} : \mathbf{A} \mapsto \mathbf{E}$$

Given two functions $f : \mathbf{A} \mapsto \mathbf{B}$ and $g : \mathbf{D} \mapsto \mathbf{B}$ such that f is a restriction of g, that is, $\mathbf{A} \subseteq \mathbf{D}$ and $f = g|_{\mathbf{A}}$, then g is an extension of f.

In this work we will only consider MMOPs whose parameter space can be *extended* to the real domain. In other words, the parameter space $\mathbf{E} = \mathbb{R}^{d_1} \times \mathbb{Z}^{d_2}$ must be a *restriction* of $\mathbb{R}^n$ for $n = d_1 + d_2$. This condition is necessary for the computation of a direction ν such that $J(x)\nu = \delta d$, since the approximation of $J(x)$ requires that samples in a small neighborhood of x can be taken (see Section 3.5). In case this sampling process is not possible, ν direction can not be computed.

Thus, the EDS method can only be used to solve MMOPs that comply with the aforementioned condition. Real variables in a mixed-integer problem are handled the same way as for MOPs, for handling integer variables the following strategy is adopted.

Let $d \in \mathbb{R}^k$ be a direction in objective space and let $x \in \mathbb{R}^{d_1} \times \mathbb{Z}^{d_2}$ be a point from which we would like to move in a direction $\nu \in \mathbb{R}^n$ such that $J(x)\nu = \delta d$ for a given step size $t \in \mathbb{R}$. For the computation of the ν direction we proceed as for the case of MOPs (Section 3.1). Nevertheless, when performing the movement along ν, i.e. $\hat{x} = x + t\nu$ we proceed as follows as in Algorithm 6.

Algorithm 6 Convert to Mixed-Integer

Input: Point $x \in \mathbb{R}^{d_1} \times \mathbb{Z}^{d_2}$, direction $\nu \in \mathbb{R}^n$, step size $t \in \mathbb{R}$ and threshold $\lambda \in (0, 1]$
Output: New point $\hat{x} \in \mathbb{R}^{d_1} \times \mathbb{Z}^{d_2}$

```
1: for Each component x_j of x do
2:     if Component x_j is discrete then
3:         if abs(ν_j) > λ then
4:             if ν_j > 0 then
5:                 x̂_j = x_j + ceil(ν_j)
6:             else
7:                 x̂_j = x_j + floor(ν_j)
8:             end if
9:         end if
10:    else
11:        x̂_j = x_j + tν_j
12:    end if
13: end for
14: return x̂
```

The main task of Algorithm 6 is to ensure that the point $\hat{x}$, which results from performing the line search along $\nu \in \mathbb{R}^n$, belongs to the appropriate space, that is, to the mixed-integer space in case the problem is mixed-integer or to the real space in case of dealing with a problem defined within the real space. The main idea of the algorithm is contained in the lines 3 to 11. An in-depth explanation of it is given next.

Recall that $\nu \in \mathbb{R}^n$ is the greediest direction in parameter space such that $J(x)\nu = \delta d$, that is, for a step size $t \in \mathbb{R}$, a movement along ν, i.e. $\hat{x} = x + t\nu$, maps to a movement $d \in \mathbb{R}^k$ in objective space. Let us assume that $x \in \mathbb{R}^{d_1} \times \mathbb{Z}^{d_2}$ (that is x is a mixed-integer variable), since $\nu \in \mathbb{R}^n$ belongs to the real space if we perform the line search as stated before, $\hat{x}$ will belong to the real space instead of belonging to a mixed-integer one. Thus, in order to keep $\hat{x}$ in the appropriate space:

- We round the value of the i-th component of ν and add it to the corresponding component of x (lines 5 and 7), if the value of such component surpasses a certain user defined threshold $\lambda \in (0, 1]$ (line 3), when dealing with integer variables.

- We proceed as usual, i.e., $\hat{x}_i = x_i + t\nu_i$, when dealing with real variables (line 11).

We will now explain the role of the threshold λ. By considering that each of the components of ν determines how much to move in each coordinate, it can be seen that the bigger the magnitude of a certain coordinate direction, the greater the reason to increase/decrease such component in the new point $\hat{x}$. Therefore, the necessity for a threshold on the magnitude of each component of the direction v arises, we have decided to call such threshold λ and although it is a user defined parameter, our experiments have shown that setting $\lambda \in [0.2, 0.4]$ usually leads to good results.

Finally, the reason for choosing *floor* and *ceil* functions over *round* is due to the following observation: *round* function is always equivalent to the result of the *floor* function if $\lambda < 0.5$, hence, if the user defines $\lambda \in (0, 0.5)$ the value of the i-th component will always be mapped to the smallest following integer. The situation is analogous for the *ceil* function when $\lambda \in [0.5, 1)$. This is of course an undesirable behavior according to the above discussion, thus, *floor* and *ceil* functions are used instead of *round* since they provide more flexibility for mapping the real components of ν to an integer space.

Hence, to make the EDS method capable of solving MMOPs a call of Algorithm 6 is necessary for every new point that has to be computed.

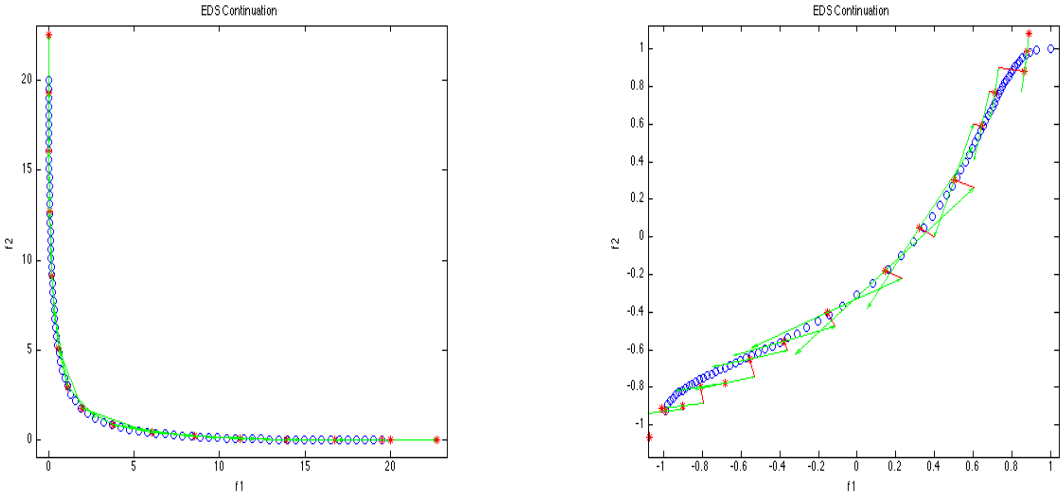
3.7 An Example of the EDS Method

In the following we would like to exemplify how the various parameters of the EDS method have an impact on the performance and the reliability of the method. For this example we will use the bi-objective function described in [35], which is defined as follows:

$$\begin{aligned} f_1(x) = &\ (x_2 - a_2)^2 + (x_1 - a_1)^4 \\ f_2(x) = &\ (x_1 - b_1)^2 + (x_2 - b_2)^4, \end{aligned} \tag{3.28}$$

for $a = (1, 1)$ and $b = -a$. This function has a convex Pareto front with a non-linear Pareto set. Figure 3.7 shows the resolution of a bi-objective problem using the EDS method, for this example we set $\tau = 2$ since our goal is to depict predictor and corrector steps. Blue dots represent the real Pareto front and set. The points computed by the EDS method are depicted in red color, green arrows represent the predictor directions while red arrows represent corrector directions. As can be observed in the picture, the predictor directions are not necessarily tangent to the Pareto set, it can

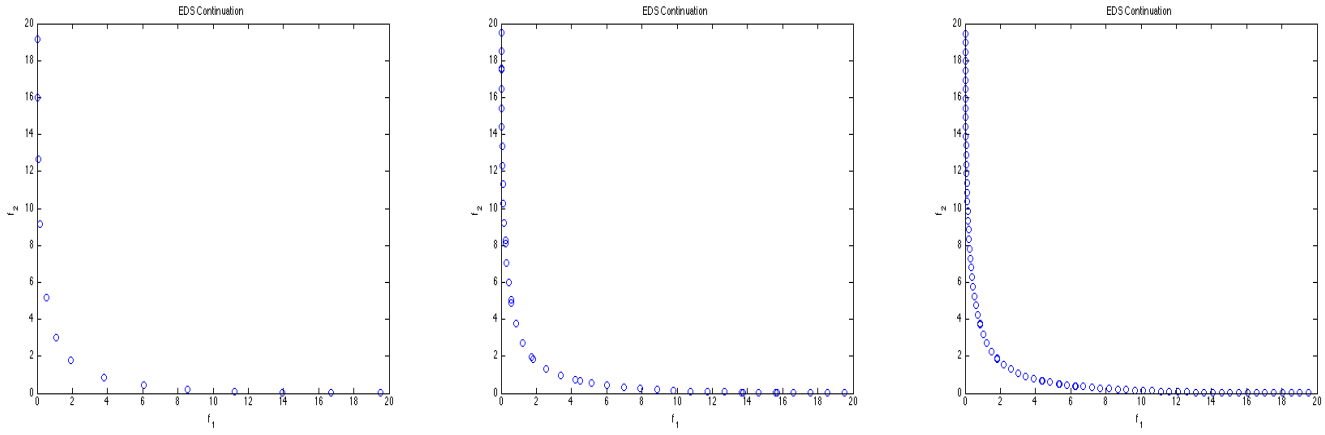
also be observed that the computed points do not necessarily lie on the Pareto set but instead on the Pareto front. Also note that two predictors are computed per point, nevertheless not all of them lead to new correctors (see Section 3.4.1).



(a) Objective space (b) Decision space

Figure 3.7: Resolution of a bi-objective problem by using the EDS method

The computed Pareto front (PF) for this example with $\tau = 3$ can be seen in Figure 3.8a, a finer discretization of the objective space can be seen in Figure 3.8b where $\tau = 1$. An even finer discretization, where $\tau = 0.5$, is shown in Figure 3.8c.



(a) Computed PF for $\tau = 3$ (b) Computed PF for $\tau = 1$ (c) Computed PF for $\tau = 0.5$

Figure 3.8: Pareto fronts computed by the EDS method for different values of τ

The number of solutions computed for each of the τ values is displayed in Table 3.1. As it can be observed the lower the value of τ the more solutions we get.

τ value	Solutions
3	14
1	44
0.5	80

Table 3.1: Number of solutions computed for the different values of τ

The plots displayed in Picture 3.8 along with the results in Table 3.1 help to have a better understanding of the τ role.

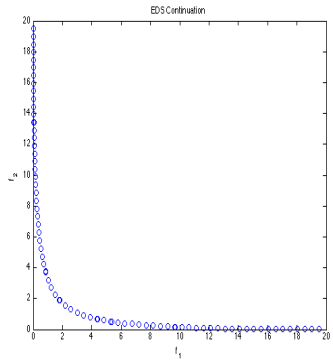
3.7.1 On the Impact of the δ Thresholds

Here we would like to demonstrate how both δ thresholds impact on the overall performance of the EDS method. For this we will consider test problem (3.28) again. We set $\tau = 0.5$ and will show some combinations of the max_δ and min_δ in order to let the reader gain a bigger understanding of the role of such thresholds. Table 3.2 displays five different settings for the δ thresholds in the max_δ and min_δ columns along with the number of solutions, number of function evaluations and the $\triangle_2$ values for each of the combinations.

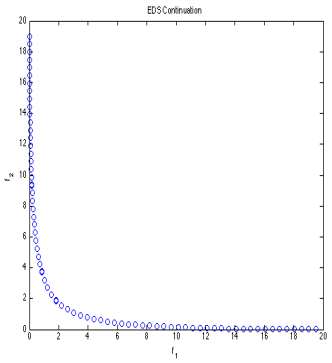
Setting	max_δ	min_δ	Solutions	Feval	$\triangle_2$	Avg. Feval/Sols
1	10^{-1}	10^{-3}	81	423	0.15	5.2
2	10^{-1}	10^{-2}	77	320	0.19	4.1
3	10^{-1}	10^{-1}	77	269	0.19	3.4
4	10^{-2}	10^{-3}	26	164	10.16	6.3
5	10^{-3}	10^{-3}	7	45	15.53	6.4

Table 3.2: Impact of the δ thresholds on the overall performance of the EDS method

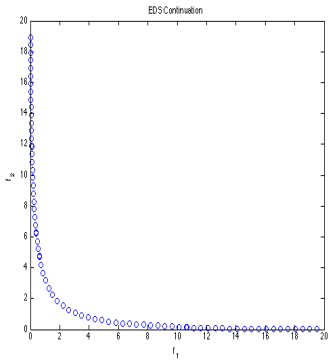
As can be observed by the results in Table 3.2 the choice of the values of the δ thresholds have a large impact on the performance of the EDS method. In some cases this impact is so large that the EDS method is unable to compute further correctors and hence stops it execution as was the case for the last two settings in Table 3.2. Also note that, as expected, the more tight these thresholds are the more function evaluations are used per computed solution. Therefore a good balance between max_δ and min_δ thresholds has to be considered in order to obtain reliable results. Pictures for each of the computed Pareto fronts for each of the settings in Table 3.2 are shown in Figure 3.9.



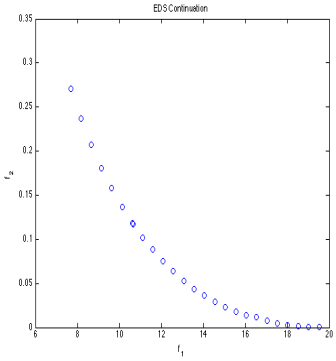
(a) Computed PF for setting 1



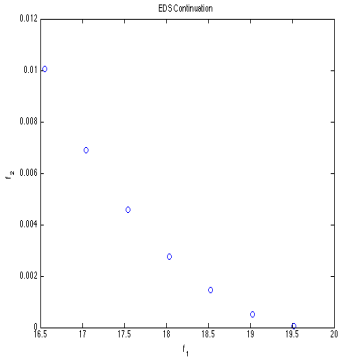
(b) Computed PF for setting 2



(c) Computed PF for setting 3



(d) Computed PF for setting 4



(e) Computed PF for setting 5

Figure 3.9: Pareto fronts computed by the EDS for the different settings of max_δ and min_δ

3.7.2 On the Impact of the Size of the Neighborhood $N(x)$

As mentioned in Section 3.5 neighboring information can be used in order to save function evaluations and, therefore, improve the performance of the EDS method. Here we would like to demonstrate, trough some examples, how the choice of the size of the neighborhood $N(x)$ impacts the performance of the EDS method.

Once again we will use function (3.28) for our examples. We set $\tau = 0.5$, max_$\delta = 10^{-1}$ and min_$\delta = 10^{-3}$. For this example we put special emphasis on the number of function evaluations, the number of neighboring solutions used in the computation and the quality of the computed solutions (measured by the $\triangle_2$ indicator). Table 3.3 summarizes the results obtained for five different neighborhood sizes.

Size of $N(x)$	Solutions	Feval	# Neighbors	$\triangle_2$	Avg. Feval/Sols
0	80	502	0	0.153	6.2
0.01	84	458	28	0.158	5.4
0.02	80	396	163	0.140	4.9
0.03	89	357	177	0.164	4
0.04	74	344	249	0.164	4.6
0.05	78	334	268	0.171	4.2

Table 3.3: Impact of the size of $N(x)$ on the overall performance of the EDS method

First of all note that the first setting is almost same as the first setting in Table 3.2 but here we increased the tolerance for smaller step sizes in the corrector, hence, this setting may have slightly more correctors than the setting in Table 3.2, this explains with this second setting uses more function evaluations the former. Note, by the data in Table 3.3, that by increasing the size of the neighborhood more points can be reused, nevertheless, the bigger the size of the neighborhood is the less accurate mapping (3.1) and hence, the more function evaluations needed to reach a new critical point by means of the corrector phase. Note the negative impact this effect has on the quality of the solution computed. This effect also explains why although the number of neighboring solutions reused increases in the last three settings, the number of function evaluations required by the method does not improve much. In an extreme case, where the size of the neighborhood is too large, the corrector phase may not be able to compute further critical points, leading to a premature termination of the EDS method.

Chapter 4

Numerical Results and Experiments

In this chapter we present the numerical results obtained with the EDS method on unconstrained, box constrained and mixed-integer problems up to three objectives. The EDS method is compared against the DZZ method [41] and NSGA-II algorithm [45]. We chose to compare against the DZZ method since it is a state-of-the-art algorithm in the field of mixed-integer multi-objective optimization. As for NSGA-II, it is one of the most widely used algorithms for solving MOPs. Although we are aware that there is no "fair" comparison between NSGA-II and EDS algorithms given that the former is a *global* optimization method while the latter is of *local* nature, we consider this comparison necessary in order to show the quality of the solutions obtained by our method. The parameter settings for each algorithm on each of the problems is presented in Appendix A.

4.1 General Setting

We will now describe the general setting of our experiments. DZZ version used in this experiment was provided by Dr. Honggang Wang from Rutgers University, being this the better version up to date. NSGA-II algorithm was downloaded from [54] which is version of NSGA-II with support for mixed-integer problems [55]. For each of the problems a similar number of solutions was computed by each algorithm in order to make a fairer comparison. For the case of NSGA-II a budget of approximately four times the number of function evaluations spent by EDS was assigned as stopping criteria. In the case of the DZZ method the First Pareto Solution (FPS) is given in order to restrict the comparison to the continuation method (Zig and Zag steps). For each of the test problems the three algorithms were run ten times and the better

solution of each of them is used for the comparison.

For each of the solved problems a table summarizing key information for the comparison is presented. We put special emphasis on the number of function evaluations used by each algorithm displayed in the *Feval* column and on the value of the $\triangle_2$ and $\triangle_3$ performance indicators [33]. A ratio of function evaluations per solution is shown on the column *Funeval/sol*.

Finally, for the computation of the $\triangle_p$ performance indicator the real Pareto front was used in all of the cases except for the last function presented in this chapter, where the Pareto front was approximated by performing ten *long* runs of the NSGA-II algorithm and keeping the non-dominated points among all of the runs. Plots for each one of the reference fronts are shown in Appendix B.

4.2 Continuous Models

First, we will show the performance of the EDS method on some well-known continuous test problems. The results obtained provide clear evidence that the EDS method can be used as a continuation method for solving continuous MOPs.

4.2.1 Binh Function

First we consider the following bi-objective problem [13]:

$$\begin{aligned} f_1(x) &= ||x - a_1||_2^2 \\ f_2(x) &= ||x - a_2||_2^2, \end{aligned} \tag{4.1}$$

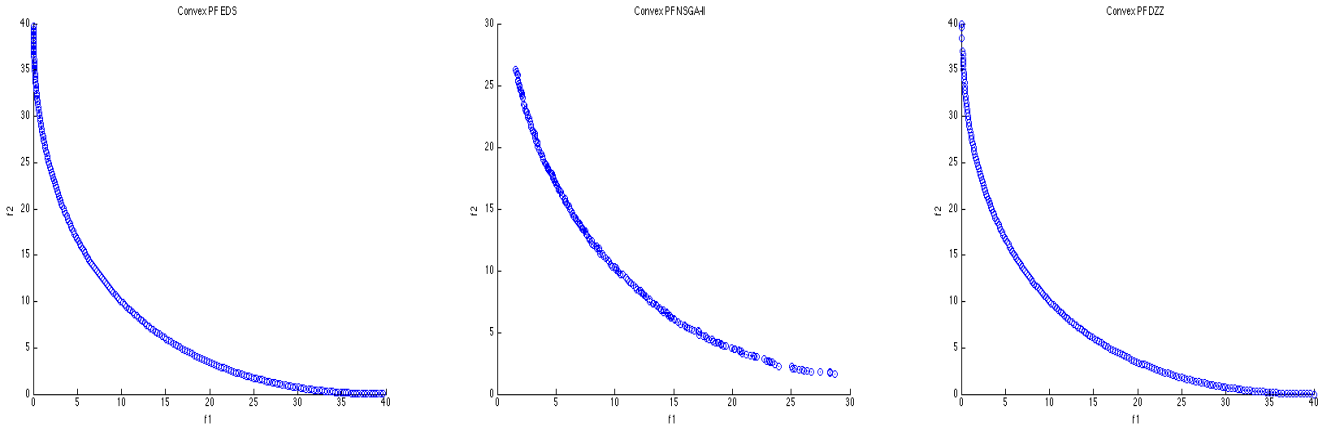
where $a_1 = (1, \ldots, 1)^T \in \mathbb{R}^n$ and $a_2 = -a_1$ for $n = 10$. This is a widely used benchmark function with a convex Pareto front and a linear Pareto set. The results obtained by EDS, NSGA-II and DZZ are summarized in Table 4.1.

Algorithm	Int. Var.	Real Var.	Sols	Feval	$\triangle_2$	$\triangle_3$	Funeval/Sol
EDS	0	10	216	1205	0.09	0.10	5.6
NSGA-II	0	10	200	5000	4.40	5.70	25
DZZ	0	10	212	3282	0.11	0.15	15.5

Table 4.1: Summarized results for Binh function

As it can be observed by the data in Table 4.1 the EDS method outperforms both NSGA-II and DZZ. In both cases the $\triangle_p$ values of EDS are better with a lower

number of function evaluations. It can be appreciated in this experiment that EDS uses in average three times less function evaluations than DZZ and five times less than NSGA-II. The quality of the Pareto fronts computed by each method can be observed in Figure 4.1.



(a) EDS (b) NSGA-II (c) DZZ

Figure 4.1: Pareto fronts of the Binh function computed by the different methods

4.2.2 Fonseca Function

Another well known benchmark function is Fonseca function [13]:

$$\begin{aligned} f_1(x) &= 1 - e^{(-(x_1-1)^2-(x_2+1)^2)} \\ f_2(x) &= 1 - e^{(-(x_1+1)^2-(x_2-1)^2)}. \end{aligned} \tag{4.2}$$

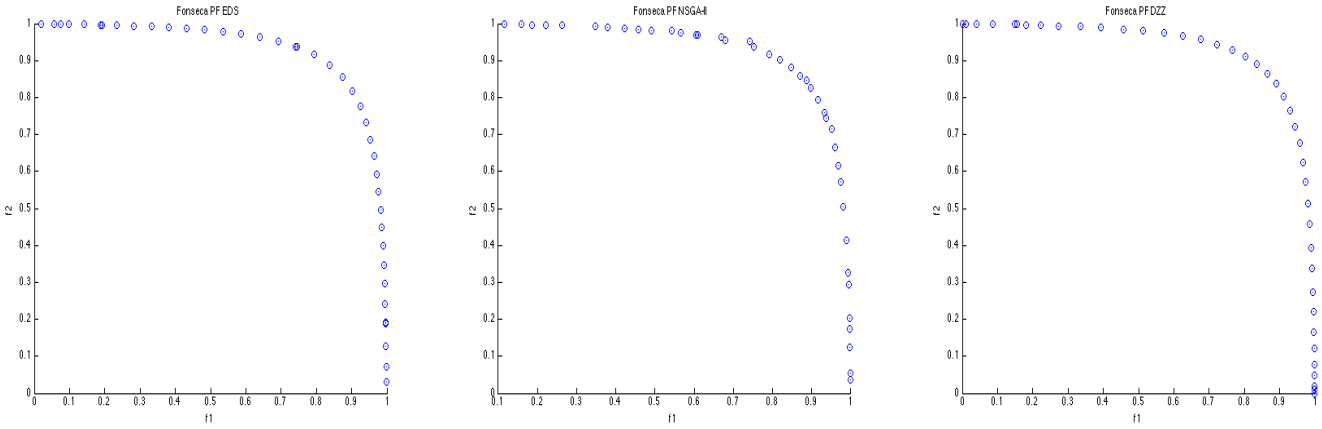
This is a bi-objective function with a linear Pareto set and a concave Pareto front. The results obtained by EDS, DZZ and NSGA-II on this function are shown in Table 4.2.

Algorithm	Int. Var.	Real Var.	Sols	Feval	$\triangle_2$	$\triangle_3$	Funeval/Sol
EDS	0	2	40	62	0.014	0.015	1.5
NSGA-II	0	2	40	240	0.033	0.045	6
DZZ	0	2	41	80	0.014	0.015	2

Table 4.2: Summarized results for Fonseca function

The results depicted in Table 4.2 show that DZZ and EDS methods obtained both a similar Pareto front being way better than the results obtained by NSGA-II.

The performance of the EDS and DZZ methods is quite similar as well, being both much more efficient than NSGA-II using up to four times less function evaluations per solution. The Pareto fronts computed by each method are shown in Figure 4.2.



(a) EDS (b) NSGA-II (c) DZZ

Figure 4.2: Pareto fronts of the Fonseca function computed by the different methods

4.2.3 Dent Function

Next is the Dent function [56] defined as:

$$f_1(x) = \frac{1}{2}(\sqrt{1+(x_1+x_2)^2} + \sqrt{1+(x_1-x_2)^2} + x_1 - x_2) + \lambda \cdot e^{-(x_1-x_2)}$$
$$f_1(x) = \frac{1}{2}(\sqrt{1+(x_1+x_2)^2} + \sqrt{1+(x_1-x_2)^2} - x_1 + x_2) + \lambda \cdot e^{-(x_1-x_2)}. \quad (4.3)$$

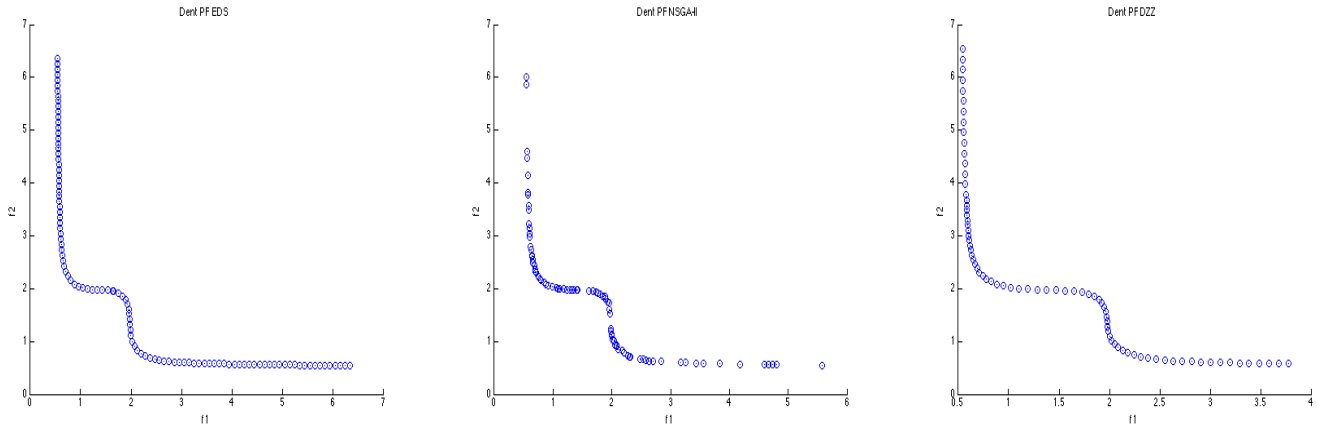
The Pareto set of this function is a line and the Pareto front is convex-concave for $\lambda = 0.85$ as chosen here. Table 4.3 shows the results obtained by EDS, DZZ and NSGA-II.

Algorithm	Int. Var.	Real Var.	Sols	Feval	$\triangle_2$	$\triangle_3$	Funeval/Sol
EDS	0	2	111	217	0.03	0.04	2
NSGA-II	0	2	110	990	0.22	0.30	9
DZZ	0	2	80	195	0.77	1.07	2.5

Table 4.3: Summarized results for Dent function

It can be observed from Table 4.3 that the EDS method is clearly superior to both DZZ and NSGA-II. Compared against DZZ, the EDS method is slightly more efficient

while delivering a Pareto front of higher quality. In fact, DZZ could not compute the entire Pareto front for this test problem. Compared against NSGA-II it can be seen that the EDS method is four times more efficient with a better solution delivered. In fact, the solution delivered by the EDS method is of high quality given its $\triangle_p$ values. Pictures for the computed Pareto fronts are shown in Figure 4.3. Note how the Pareto front computed by DZZ algorithm is incomplete.



(a) EDS (b) NSGA-II (c) DZZ

Figure 4.3: Pareto fronts of the Dent function computed by the different methods

4.2.4 ZDT1 Function

Now we test on ZDT1 function [57] defined as follows:

$$\begin{aligned} f_1(x) = &\quad x_1 \\ f_2(x) = &\quad g(x) \cdot \left(1 - \sqrt{\frac{f_1(x)}{g(x)}}\right) \\ g(x) = &\quad 1 + \frac{9}{n-1}\sum_{i=1}^{n} x_i \\ s.t. &\quad 0 \leq x_i \leq 1, \quad i = 1, \ldots, n. \end{aligned} \tag{4.4}$$

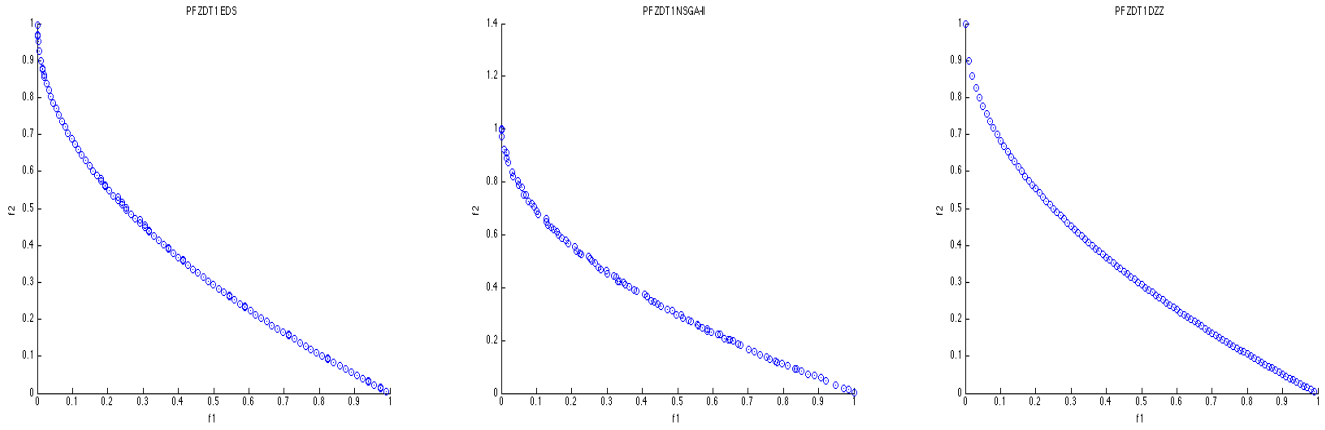
where $n = 10$. This function has a convex Pareto front and also has box constraints, therefore, this example also demonstrates how the EDS method can handle box constrained functions. The results for the three methods are displayed in Table 4.4.

By the results in Table 4.4 it can be observed that the DZZ method performed

Algorithm	Int. Var.	Real Var.	Sols	Feval	$\triangle_2$	$\triangle_3$	Funeval/Sol
EDS	0	10	99	1123	0.0051	0.0052	11
NSGA-II	0	10	100	4000	0.0073	0.0075	40
DZZ	0	10	101	1100	0.0052	0.0073	10

Table 4.4: Summarized results for ZDT1 function

slightly better than the EDS method for this test function, using in average one function evaluation less per solution computed. It can also be observed that the $\triangle_p$ values of both methods are quite similar. Finally, it can be seen that EDS and DZZ methods outperformed NSGA-II algorithm, being much more efficient and delivering better results. The Pareto fronts obtained by each method are shown in Figure 4.4.



(a) EDS (b) NSGA-II (c) PDZZ

Figure 4.4: Pareto fronts of the ZDT1 function computed by the different methods

4.2.5 Binh tri-objective Function

We will now consider tri-objective problems. Our first experiment is conducted on an extended version of the bi-objective Binh function and is defined as follows:

$$\begin{aligned} f_1(x) &= ||x - a_1||_2^2 \\ f_2(x) &= ||x - a_2||_2^2 \\ f_3(x) &= ||x - a_3||_2^2, \end{aligned} \tag{4.5}$$

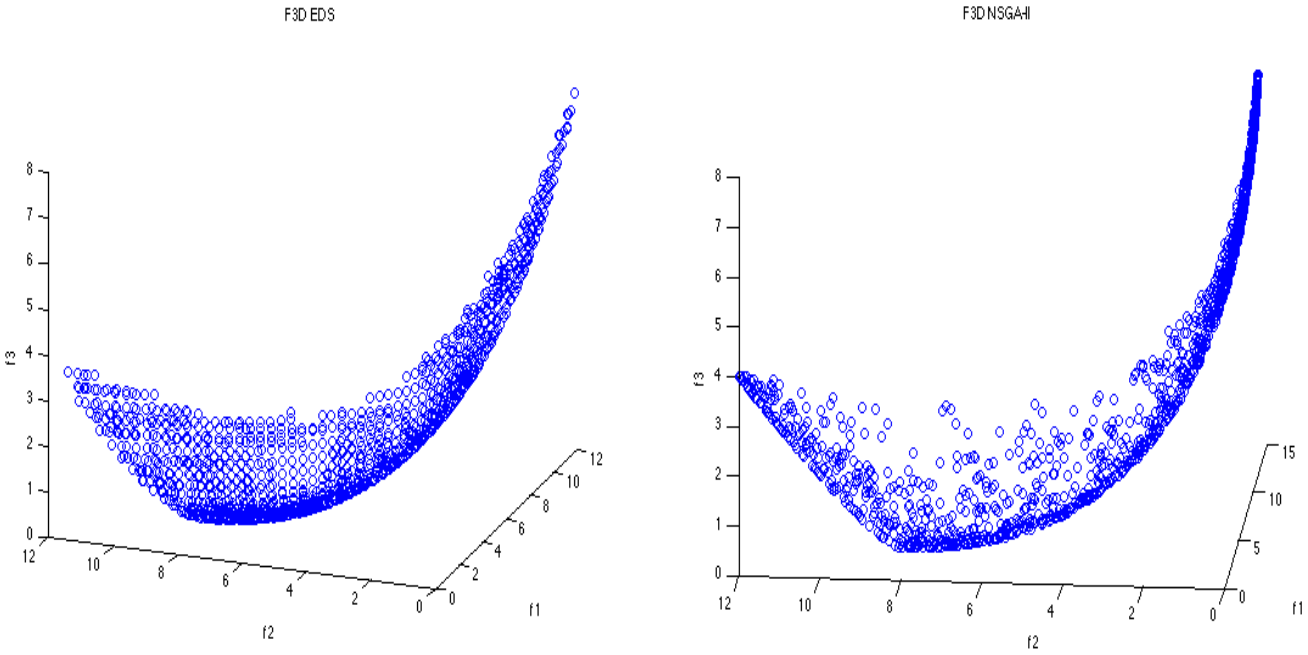
where $a_1 = (20, \ldots, 20) \in \mathbb{R}^n$, $a_2 = -a_1$ and $a_3 = (\underbrace{20, \ldots, 20}_{\text{m times}}, \underbrace{-20, \ldots, -20}_{\text{n-m times}}) \in \mathbb{R}^n$ for

$n = 3$ and $m = ceil(n/2)$. This function has a convex Pareto front. No comparison against DZZ is possible for tri-objective problems since the method is only developed for bi-objective problems, thus, only the comparison against NSGA-II is presented in Table 4.5.

Algorithm	Int. Var.	Real Var.	Sols	Feval	$\triangle_2$	$\triangle_3$	Funeval/Sol
EDS	0	3	1267	11567	0.102	0.106	9
NSGA-II	0	3	1200	45600	0.145	0.149	38

Table 4.5: Summarized results for Binh3 function

As can be observed by the results in Table 4.5 the EDS method performs way better than NSGA-II being more accurate in the quality of the solutions delivered with approximately four times less function evaluations. The fronts computed by both methods are shown in Figure 4.5.



(a) EDS (b) NSGA-II

Figure 4.5: Pareto fronts of the Binh3 function computed by the different methods

It can be seen on the pictures that the solutions obtained by the EDS method are more evenly distributed that the ones obtained by NSGA-II algorithm.

4.2.6 DTZL1 Function

The DTLZ benchmark functions [13] are scalable in the number of variables and objectives. We will first test on DTLZ1 function, which is a multimodal function with box constraints. It is defined as follows:

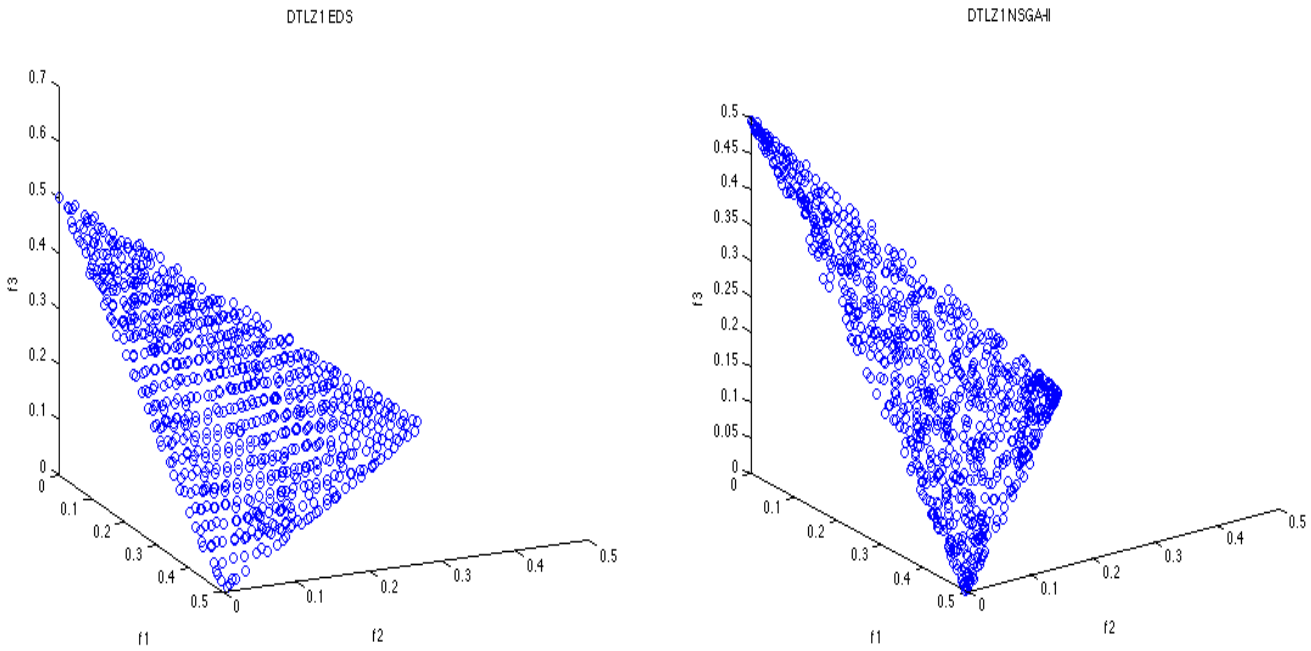
$$\begin{aligned} f_1(x) = &\ \frac{1}{2}x_1x_2(1+g(x)) \\ f_2(x) = &\ \frac{1}{2}x_1(1-x_2)(1+g(x)) \\ f_3(x) = &\ \frac{1}{2}(1-x_1)(1+g(x)) \\ g(x) = &\ 100(10+\sum_{i=3}^{n}(x_i-0.5)^2-\cos(20\pi(x_i-0.5))) \\ s.t. \quad &\ 0 \leq x_i \leq 1, \quad i=1,\ldots,n. \end{aligned} \tag{4.6}$$

For this example we set $n = 3$. As in the previous example DZZ is not applicable to this function, hence the comparison is only made between the EDS method and NSGA-II algorithm. Results are shown in Table 4.6.

Algorithm	Int. Var.	Real Var.	Sols	Feval	$\triangle_2$	$\triangle_3$	Funeval/Sol
EDS	0	3	870	5991	0.0085	0.0082	7
NSGA-II	0	3	900	23400	0.0093	0.0094	26

Table 4.6: Summarized results for DTLZ1 function

Once again, as indicated by the results in Table 4.6 the EDS method performs better than the NSGA-II algorithm, despite the latter uses about four times more function evaluations. Figure 4.6 shows the Pareto fronts computed by both methods.



(a) EDS (b) NSGA-II

Figure 4.6: Pareto fronts of the DTLZ2 function computed by the different methods

4.2.7 DTLZ2 Function

We will now test on DTLZ2 which is a box constrained k-objective function. For the purposes of this demonstration we restrict it to three objectives. Its definition is as follows:

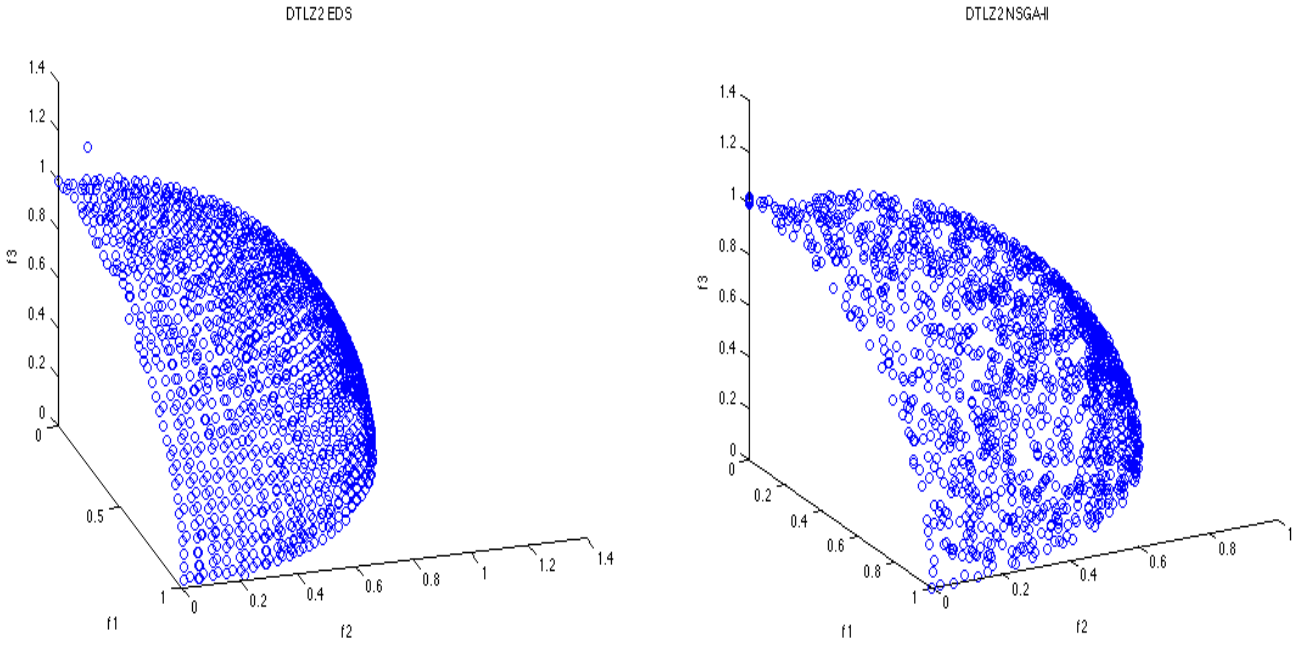
$$\begin{aligned} f_1(x) = &\quad cos(\frac{\pi}{2}x_1)\ldots\cos(\frac{\pi}{2}x_{k-1})(1+g(x)) \\ f_2(x) = &\quad cos(\frac{\pi}{2}x_1)\ldots\sin(\frac{\pi}{2}x_{k-1})(1+g(x)) \\ f_3(x) = &\quad sin(\frac{\pi}{2}x_1)(1+g(x)) \\ g(x) = &\quad \sum_{i=k}^{n}(x_i-0.5)^2 \\ s.t. &\quad 0 \leq x_i \leq 1, \quad i=1,\ldots,n. \end{aligned} \tag{4.7}$$

For this example we set $n = 3$. Once again, the DZZ method is not applicable to this kind of functions. Hence the comparison shown in Table 4.7 considers only EDS and NSGA-II algorithms.

Algorithm	Int. Var.	Real Var.	Sols	Feval	$\triangle_2$	$\triangle_3$	Funeval/Sol
EDS	0	3	1538	6375	0.152	0.157	4
NSGA-II	0	3	1500	25500	0.1861	0.1863	17

Table 4.7: Summarized results for DTLZ2 function

The above results show that the overall performance of the EDS method is better than the one of NSGA-II. It is important to recall that in order to make the comparison between the EDS method and NSGA-II algorithm as fair as possible a budget of four times the number of functions evaluations spent by EDS was assigned to NSGA-II. Pictures of the Pareto fronts obtained are shown in Figure 4.7.



(a) EDS (b) NSGA-II

Figure 4.7: Pareto fronts of the DTLZ2 function computed by the different methods

4.3 Integer Models

In this section we will present some integer problems solved by the EDS method. As for the case of continuous problems, a comparison between EDS, DZZ and NSGA-II algorithms is presented for each test problem.

4.3.1 Binh Integer Function

This function is based on the previously introduced Binh function but considering an integer search space. Pareto front and Pareto set remain the same, the formal definition of the function is:

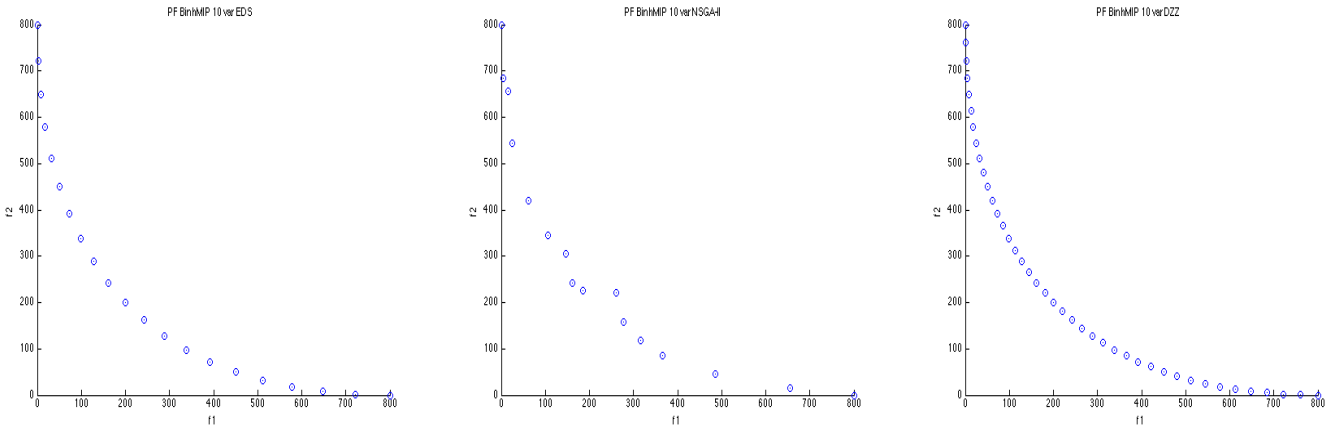
$$\begin{aligned} f_1(x) &= ||x - a_1||_2^2 \\ f_2(x) &= ||x - a_2||_2^2, \end{aligned} \tag{4.8}$$

where $a_1 = (1, \ldots, 1)^T \in \mathbb{R}^n$ and $a_2 = -a_1$ for $n = 10$ and $x \in \mathbb{Z}^n$. The results obtained by EDS, NSGA-II and DZZ are summarized in Table 4.8.

Algorithm	Int. Var.	Real Var.	Sols	Feval	$\triangle_2$	$\triangle_3$	Funeval/Sol
EDS	0	10	21	240	29.26	42.35	11.5
NSGA-II	0	10	20	1000	39.09	49.20	50
DZZ	0	10	41	529	17.28	39.09	13

Table 4.8: Summarized results for Binh Integer function

As can be observed by the results in Table 4.8, EDS and DZZ methods display similar results, being the EDS method slightly better in terms of function evaluations while the DZZ method is better in terms of the $\triangle_p$ performance indicator. The DZZ was able to compute as twice points as the EDS method, this explains the better values of the $\triangle_p$ indicator for the DZZ results. As for the NSGA-II algorithm, it can be observed that both EDS and DZZ methods have a better overall performance. Pictures of the fronts computed by each method are shown in Figure 4.8. Note that both EDS and DZZ methods computed smooth fronts.



(a) EDS (b) NSGA-II (c) DZZ

Figure 4.8: Pareto fronts of the Convex integer function computed by the different methods

4.3.2 Dent Integer Function

Next, we test on a discrete version of the Dent function previously introduced. The definition of this function is:

$$f_1(x) = \frac{1}{2}(\sqrt{1+(x_1+x_2)^2} + \sqrt{1+(x_1-x_2)^2} + x_1 - x_2) + \lambda \cdot e^{-(x_1-x_2)}$$
$$f_1(x) = \frac{1}{2}(\sqrt{1+(x_1+x_2)^2} + \sqrt{1+(x_1-x_2)^2} - x_1 + x_2) + \lambda \cdot e^{-(x_1-x_2)}. \quad (4.9)$$

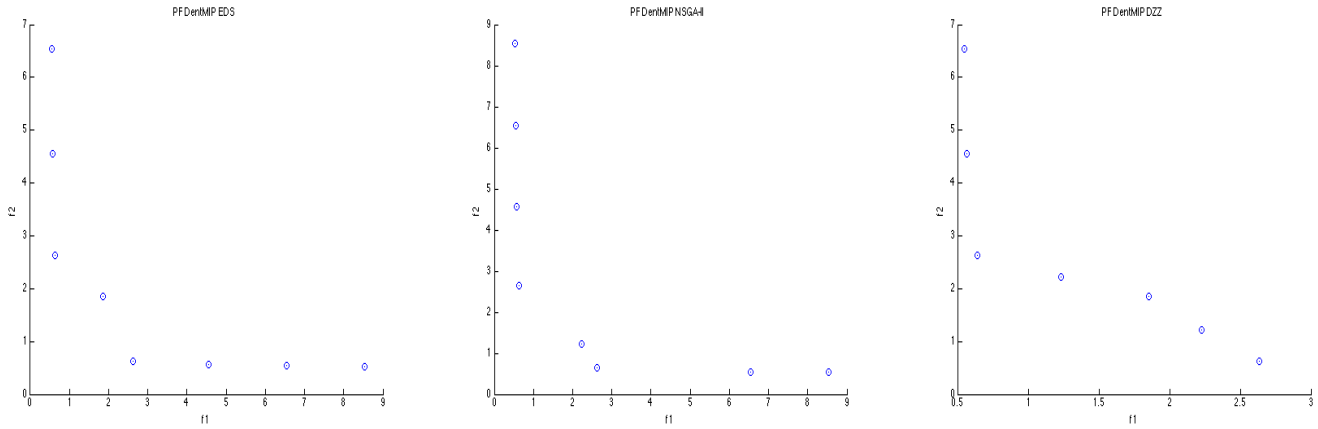
The Pareto set of this function is a line and the Pareto front is convex-concave for $\lambda = 0.85$ as chosen here. We use the same setting as for the continuous case but setting $x \in \mathbb{Z}^n$. Table 4.9 shows the results obtained by EDS, DZZ and NSGA-II.

Algorithm	Int. Var.	Real Var.	Sols	Feval	$\triangle_2$	$\triangle_3$	Funeval/Sol
EDS	0	2	8	35	0.70	0.99	4.5
NSGA-II	0	2	20	140	1.11	1.33	14
DZZ	0	2	7	19	1.64	2.12	3

Table 4.9: Summarized results for Dent integer function

From the results in Table 4.9 it is observable that the EDS method performed better than the DZZ and NSGA-II methods. In fact, the DZZ method was not able

to compute the entire Pareto front for this problem, which can be observed, along with the Pareto fronts of EDS and NSGA-II algorithms, in Figure 4.9.



(a) EDS (b) NSGA-II (c) DZZ

Figure 4.9: Pareto fronts of the Dent integer function computed by the different methods

4.3.3 ZDT1 Integer Function

Now we test on a discretized version of the ZDT1 function which proposed by Dr. Wang for his experiments with the DZZ method on [41]. We will use the same setting as the one proposed in [41], the function is defined in the following way:

$$\begin{aligned} f_1(x) = &\quad \frac{x_1}{100} \\ f_2(x) = &\quad g(x)\left(1 - \sqrt{\frac{f_1(x)}{g(x)}}\right) \\ g(x) = &\quad 1 + 9 \cdot \frac{x_2 - {x_1}^2}{100} \\ s.t. &\quad 0 \leq x_i \leq 100, \quad i = 1, \ldots, n. \end{aligned} \tag{4.10}$$

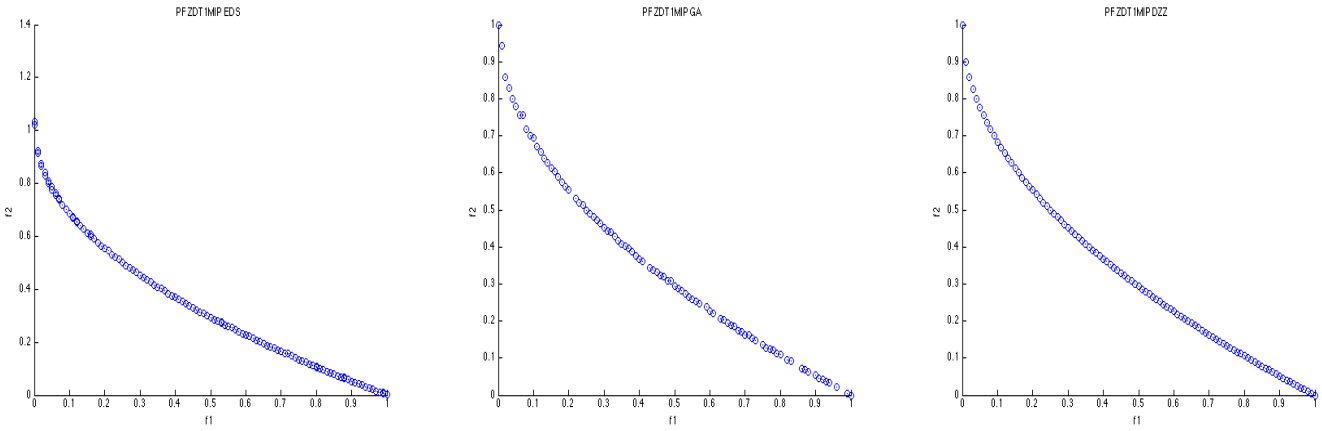
where $n = 2$ and $x \in \mathbb{Z}^n$. This function has a convex Pareto front, the solution set is contained within the domain $[0, 100] \times [0, 100]$. The results for the three methods are displayed in Table 4.10.

It can be observed by the results presented in Table 4.10 that the three methods computed good solutions, being the $\triangle_p$ values of the three quite similar. Nevertheless,

Algorithm	Int. Var.	Real Var.	Sols	Feval	$\triangle_2$	$\triangle_3$	Funeval/Sol
EDS	0	2	122	268	0.0168	0.0324	2
NSGA-II	0	2	120	1080	0.0165	0.0295	9
DZZ	0	2	100	400	0.0100	0.0214	4

Table 4.10: Summarized results for ZDT1 Integer function

it can also be observed that the EDS method was the most efficient of the three methods, using up to four times less function evaluations than NSGA-II and half the number of function evaluations of the DZZ method. The Pareto fronts obtained by each method are shown in Figure 4.10.



(a) EDS (b) NSGA-II (c) DZZ

Figure 4.10: Pareto fronts of the ZDT1 integer function computed by the different methods

4.3.4 ZDT2 Integer Function

Our next test is conducted on a discretized version of the ZDT2 function proposed in [41]. Once again we will use the same setting as the one proposed in [41]. The function is defined in the as follows:

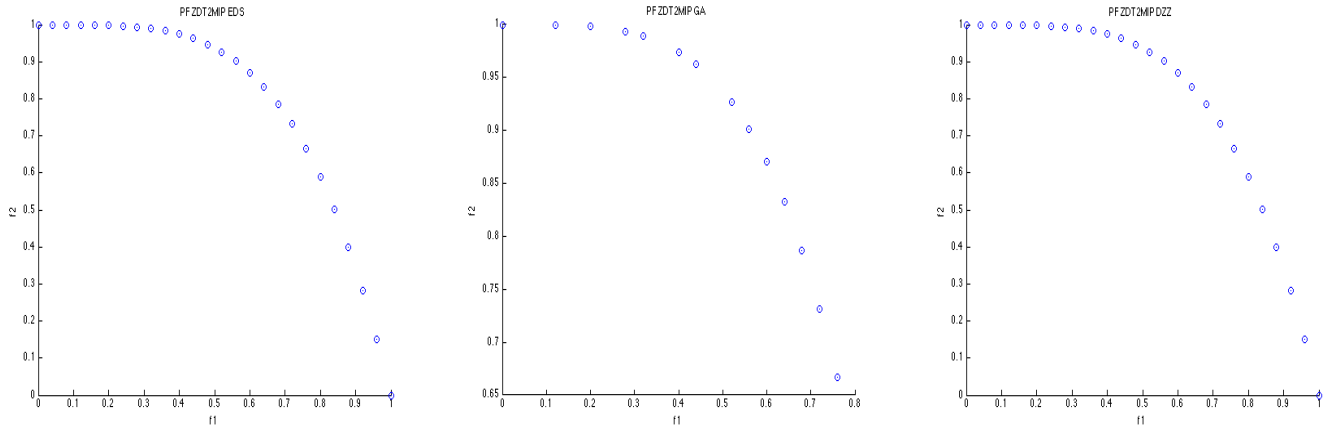
$$\begin{aligned} f_1(x) = &\quad \frac{x_1}{100} \\ f_2(x) = &\quad g(x) \cdot \left(1 - \sqrt{\frac{f_1(x)}{g(x)}}\right)^2 \\ g(x) = &\quad 1 + \left(\frac{x_2}{100}\right)^{\frac{1}{4}} \\ s.t. &\quad 0 \leq x_i \leq 100, \quad i = 1, \ldots, n. \end{aligned} \tag{4.11}$$

where $n = 2$ and $x \in \mathbb{Z}^n$. This function has a concave Pareto front, the solution set is contained within the square $[0, 100] \times [0, 100]$. The results for the three methods are displayed in Table 4.11.

Algorithm	Int. Var.	Real Var.	Sols	Feval	$\triangle_2$	$\triangle_3$	Funeval/Sol
EDS	0	2	24	49	0.044	0.065	2
NSGA-II	0	2	20	200	0.15	0.21	10
DZZ	0	2	26	106	0.031	0.051	4

Table 4.11: Summarized results for ZDT2 Integer function

The results presented in Table 4.11 demonstrate, once again, that the results obtained by the EDS and the DZZ methods are similar with respect to the $\triangle_p$ indicator. Nevertheless, the EDS method uses half of the function evaluations that the DZZ method uses. NSGA-II is again outperformed by both methods. The Pareto fronts obtained by each method are shown in Figure 4.11.



(a) EDS (b) NSGA-II (c) DZZ

Figure 4.11: Pareto fronts of the ZDT2 Integer function computed by the different methods

4.4 A Mixed-Integer Model

Finally, in this section we present one mixed-integer problem solved by the EDS method. A comparison between EDS and NSGA-II algorithms is presented. No comparison against the DZZ method is possible since this is a tri-objective problem. This test problem is a modified version of Problem (4.6) but now considering some variables integer and some real. The shape of the Pareto front is similar to the one of Function (4.6). Its definition is as follows:

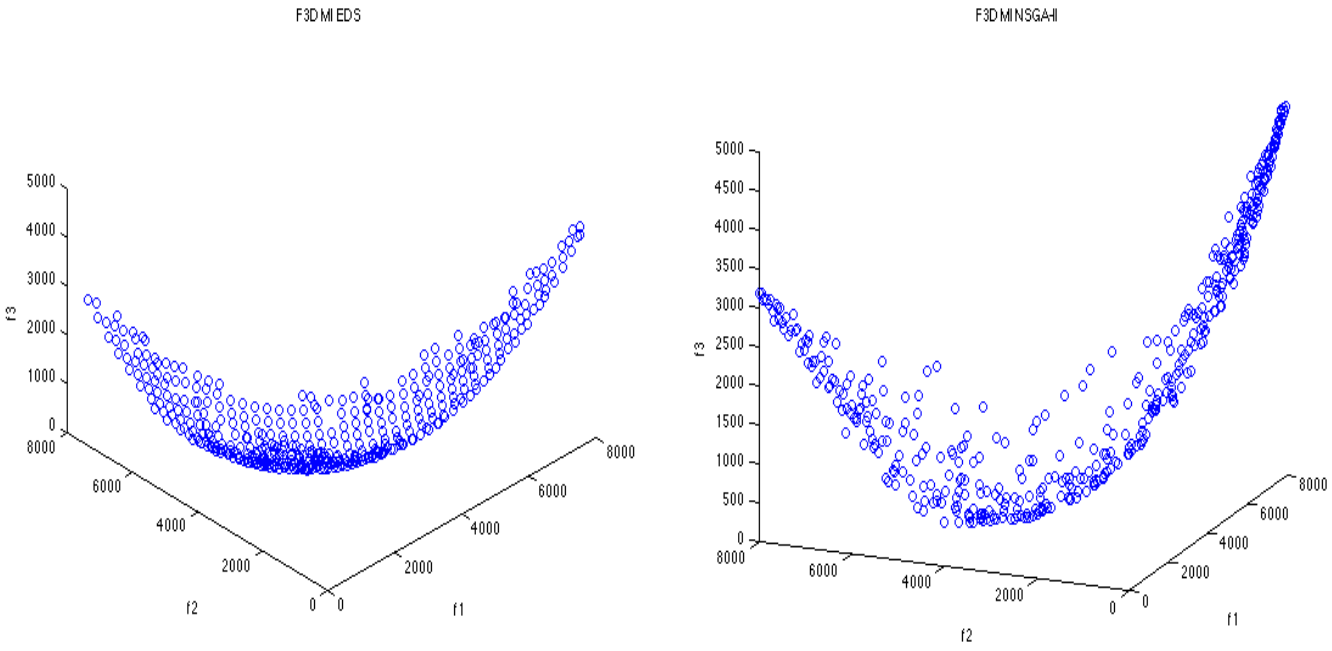
$$\begin{aligned} f_1(x) &= ||x - a_1||_2^2 \\ f_2(x) &= ||x - a_2||_2^2 \\ f_3(x) &= ||x - a_3||_2^2, \end{aligned} \tag{4.12}$$

for $a_1 = (20, \ldots, 20) \in \mathbb{R}^d$, $a_2 = -a_1$ and $a_3 = (\underbrace{20, \ldots, 20}_{\text{m times}}, \underbrace{-20, \ldots, -20}_{\text{n-m times}}) \in \mathbb{R}^d$ for $m = \lceil (\rceil d/2)$, $d = d_1 + d_2$, $d_1 = 3$, $d_2 = 2$, and $x \in \mathbb{R}^{d_1} \times \mathbb{Z}^{d_2}$. For this particular function $x_1, x_2 and x_3$ are real valued and x_4, x_5 are integer valued. This function has a convex Pareto front. Since comparison against DZZ is not possible, only the comparison against NSGA-II is presented. The results of such comparison are shown in Table 4.12.

Algorithm	Int. Var.	Real Var.	Sols	Feval	$\triangle_2$	$\triangle_3$	Funeval/Sol
EDS	2	3	434	6995	135.88	150.74	9
NSGA-II	2	3	450	27000	136.69	145.36	38

Table 4.12: Summarized results for Binh3 MI function

As the data in Table 4.12 shows, both methods, the EDS method and the NSGA-II algorithm deliver similar quality solutions. Nevertheless it is important to note that the EDS method used four times less function evaluations than NSGA-II to reach the same quality of solutions. The high values on the $\triangle_p$ values of each method are due to the scale of the objective space, going up to 8000 units in one of the objective functions. The fronts computed by both methods are shown in Figure 4.12.



(a) EDS

(b) NSGA-II

Figure 4.12: Pareto fronts of the Binh3 MI function computed by the different methods

It can be observed in the above pictures that the EDS method had some difficulties computing the narrow parts of the Pareto front, nevertheless, it can also be observed that the solutions of the EDS method are more evenly distributed than those computed by the NSGA-II. This explains why the $\triangle_p$ values of both methods are similar. The higher values of the $\triangle_3$ indicator on the EDS method are due to the fact that as p increases in the p-norm, outliers are penalized more (see Section 2.6.4 in page 24).

Chapter 5

Conclusions and Future Work

This chapter summarizes the thesis work, discusses its findings and contributions and points out its limitations. In a second part of this chapter directions for future research are outlined.

5.1 Obtained Results

Here we will briefly discuss the main results obtained by this thesis work.

First of all, the EDS method has a better, more reliable mechanism for determining critical points than the one in the DS method, namely the δ threshold (see Section 3.3.2 in page 35). By defining suitable boundaries (which are problem dependent) on the value of δ the EDS corrector is capable of approximating critical points as described in Section 3.3.2 in page 35. Of course, the boundaries on the δ threshold will define the quality of the computed solutions, small values for the δ thresholds may improve the quality of the computed solutions but may also make the whole process more costly in terms of function evaluations or may even make infeasible for the corrector phase to compute a critical point. On the other hand, loose boundaries may lead to the computations of suboptimal or even dominated solutions. Thus, some effort may be necessary when defining the values for the δ thresholds.

In addition to the mechanism for determining critical points, the EDS method has an improved efficiency over the classical DS method. This is achieved by using neighboring information for the computation of the Jacobian matrix $J(x)$. Such strategy helps EDS method to use less function evaluations during its computations. Furthermore, the EDS method can also be used for solving problems with more than two objectives. This feature was desirable since this was one of the major drawbacks of the DZZ which can only be used for solving bi-objective problems. By

making use of the steering properties of the EDS method and using the data structure described in Section 3.4.1 in page 37 the EDS method can perform this task in an efficient an easy to implement manner. Together, this two features help make the EDS method a reliable and robust continuation method for the computation of connected components of the Pareto front.

Finally, and perhaps the most important contribution of this thesis work is the capability of the EDS method for dealing with MMOPs. This is achieved by making use of the rounding strategy described in Section 3.6 in page 45.

5.2 Conclusions

From the results mentioned in Section 5.1 we can make the following discussion.

The EDS method is indeed a better version of the classical EDS method. Since, as mentioned in the last section, the EDS method incorporates the whole functionality of the DS method but has several improvements over it. For instance, a more reliable mechanism for the computation of the critical points and the use of neighboring information. It is this set of features that make this method an *enhanced* version of the DS method.

With respect to the comparisson between the EDS method and its closest competitor the DZZ method, experiments in Section 4 in page 53 help us demonstrate that the former performs way better than the latter. Firstly because the EDS method is capable of solving multi-objective problems with more than two objectives while DZZ is only capable of solving bi-objective problems. Second, because as the experiments demonstrated, the EDS method obtained better results than the DZZ in most of the test functions. This gives clear evidence that the EDS can be applied to problems that can be solved by the DZZ and obtain as good results as it and furthermore, that the EDS method can be applied for solving problems that are impossible to solve for the DZZ method.

Finally, we would like to point out that, according to this discussion, our main objective, namely the development of a novel continuation method for the treatment of MMOPs, was accomplished. The Enhanced Directed Search (EDS), as we call our method, follows the ideas proposed by the original Directed Search (DS) method [22]. Nevertheless, as indicated by its name, the EDS method has several improvements over the classical DS method.

All in all, we strongly believe that the EDS method has some serious potential for real-world applications, nevertheless, further experiments and some additional features are needed in order to guarantee its success when dealing with real-world problems. Such features and improvements are discussed in the next section.

5.3 Future Work

Several improvements and lines of research arise from the development of the EDS method. One of our major concerns for the future deals with the corrector step. Although the current corrector works properly, we believe that some other techniques, e.g., the application of classical continuation methods for tracing the solution curve of the zero set that leads to a critical point as in [22] and [53], are worth the attempt. Fortunately, modifying the corrector would have no impact in the rest of the components of the EDS method, since the corrector phase is a task completely decoupled from the rest of the algorithm.

So far, the EDS method heavily relies on four user defined parameters, i.e., the min_δ and max_δ thresholds (see Section 3.3.2 in page 35), the λ threshold (see Section 3.6 in page 45) and the size of neighborhood $N(x)$ (see Section 3.5 in page 42). We are aware that the tuning of such parameter may become a time consuming task. Therefore, further analysis on each of the aforementioned parameters is required on order to either define tight boundaries for the values of each parameter or get rid of them making EDS a much more user friendly method. For instance, further analysis on the δ thresholds may lead to the definition of suitable values for both thresholds.

A detailed analysis of the convergence rate and asymptotic time complexity of the EDS method is also encouraged for a better comprehension of the strengths and weaknesses of our proposals. The convergence analysis should determine whether if given a predictor point, and a direction pointing towards the Pareto front, will the corrector phase be always successful in finding a critical point? Furthermore, how many function evaluations may be needed in order to get such critical point? Although the experiments conducted in this thesis project may give an insight of the answer of such questions a formal framework that supports our claims is still needed.

The treatment of constrained problems is equally interesting. While some ideas are being currently explored, they are left out of the scope of this thesis project. The handling of constraints is of great importance for any method, since many of the real-world problems usually consider at least one constraint. It is therefore important to develop techniques that allow the EDS method to handle constrained problems. In particular the use of slack variables [36] is worth to explore.

Another interesting topic that arises from the development of the EDS method is that of parallelization. Given the way predictors and correctors are computed (Sections 3.2 in page 29 and 3.3 in page 31) and thanks to the successfully implementation of the data structure proposed in [53] we believe that parallelizing the computation of predictors and correctors is an attainable task. A parallel implementation of the EDS method should boost the speed and the overall performance of it, allowing it to deal in a more efficient way with problems whose function evaluation time is high.

Finally, we would like to stress that the EDS method (as all continuation methods) is of local nature. It is thus conceivable to hybridize the algorithm with a global strategy such as specialized MOEAs in order to obtain a fast and reliable procedure. Furthermore, the use of neighboring information can be exploited more with the use of memetic strategies where the solutions computed by the MOEA can be reused. So far we have only tested our method on academic examples, but the true purpose of this research is to apply the new knowledge to real-world applications. We expect that the *hybridized* approach has more probabilities of success in this environment. Regarding this matter, a starting point for hybridization can be the approach taken in [22].

Appendices

Appendix A

Parameter Setting for the Experiments

In this appendix we present the *parameter setting* used for the EDS, DZZ and NSGA-II algorithms in Chapter 4. A table displaying the values of the parameters for each algorithm is displayed for each of the test problems.

A.1 Parameter Setting of the DZZ Method

Here we present the parameter setting of the DZZ method for the experiments with the test functions in Chapter 4. The $N(x)$-zig denotes the size of the neighborhood for the zig step while the $N(x)$-zag denotes the size of the neighborhood for the zag step. For further details please see Section 2.4.

Function	$N(x)$-zig	$N(x)$-zag
Binh	0.5	0.5
Fonseca	0.1	0.1
Dent	0.1	0.1
ZDT1	0.1	0.1
Binh Int	1	1
Dent Int	1	1
ZDT1 Int	1	1
ZDT2 Int	1	1

Table A.1: Parameter setting of the DZZ method for the test functions

A.2 Parameter Setting of the EDS Method

Here we present the parameter setting of the EDS method for the experiments with the test functions in Chapter 4. The τ value indicates the spreadness of the solutions, the *epsilon*-tolerance is the deviation tolerance used for the computation of the corrector, the min_δ and max_δ thresholds are the values used as boundaries for δ in order to determine new Pareto critical points, α-tolerance is used to determine when to stop computing predictors from a particular point, $N(x)$ is the size of the neighborhood used to approximate the Jacobian matrix, finally, the λ-tolerance is used as a lower bound of when to round the value of a component during the conversion to integer step. For further details on each parameter see Chapter 3.

Function	τ	ϵ-tolerance	min_δ	max_δ	α-tolerance	$N(x)$	λ-tolerance
Binh	1	0.9	10^{-2}	10^{-1}	10^{-5}	0.1	1
Fonseca	0.05	0.9	10^{-4}	10^{-2}	10^{-5}	0.01	1
Dent	0.5	0.9	10^{-4}	10^{-2}	$7.5 * 10^{-3}$	0.1	1
ZDT1	0.05	0.5	10^{-1}	1	10^{-5}	0.01	1
Binh tri	0.3	0.8	10^{-3}	10^{-1}	10^{-2}	0.1	1
DTLZ1	0.03	0.9	10^{-3}	10^{-1}	10^{-2}	0.05	1
DTLZ2	0.05	0.9	10^{-3}	10^{-1}	10^{-2}	0.05	1
Binh Int	1	0.9	10^{-2}	10^{-1}	10^{-5}	1	0.5
Dent Int	1	0.9	10^{-2}	10^{-1}	$7.5 * 10^{-3}$	1	0.5
ZDT1 Int	0.01	0.3	$5 * 10^{-6}$	10^{-4}	10^{-5}	3	0.5
ZDT2 Int	0.1	0.3	10^{-3}	10^{-1}	10^{-3}	1	0.5
Binh tri MI	300	0	12	10^{-1}	10	3	0

Table A.2: Parameter setting of the EDS method for the test functions

A.3 Parameter Setting of the NSGA-II Algorithm

As for the NSGA-II the following values were used for all of the problems:

- Crossover fraction: 2/numVar
- Mutation fraction: 2/numVar

where numVar is the number of variables in the problem. This was done this way since our aim was just to demonstrate that the EDS method can obtain as good solutions as the NSGA-II algorithm. The rest of the parameters are taken from the setting in [55].

Appendix B

Reference Pareto Fronts of the Test Problems

Here we present the *reference* Pareto fronts of the test functions used in Chapter 4. This fronts were used as reference fronts for the computation of the $\triangle_p$ performance indicator (see Section 2.6). They are discretizations of the real Pareto fronts for each problem.

B.1 Binh Function

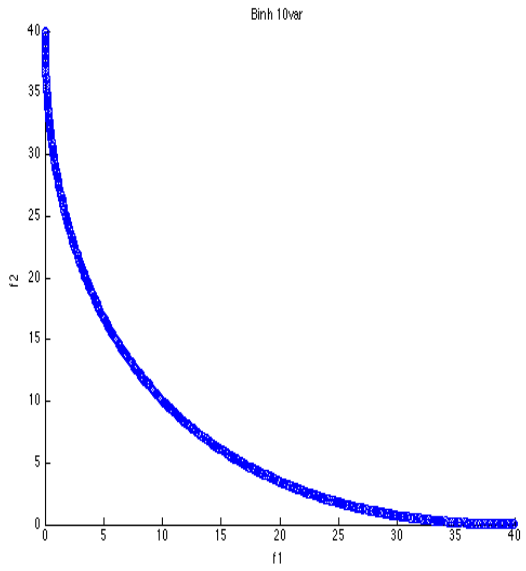


Figure B.1: Real Pareto front of the Binh function

B.2 Fonseca Function

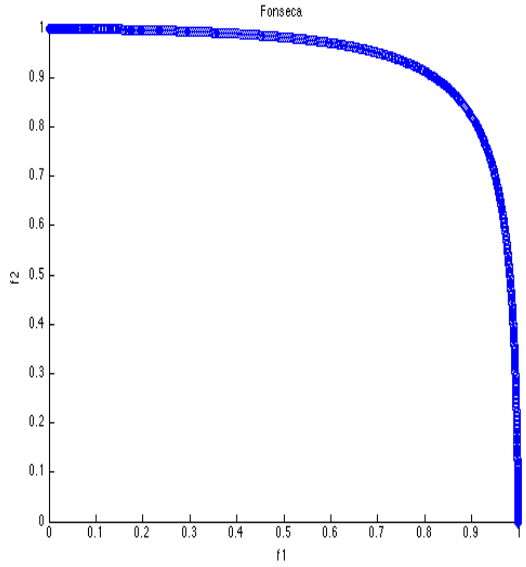


Figure B.2: Real Pareto front of the Fonseca function

B.3 Dent function

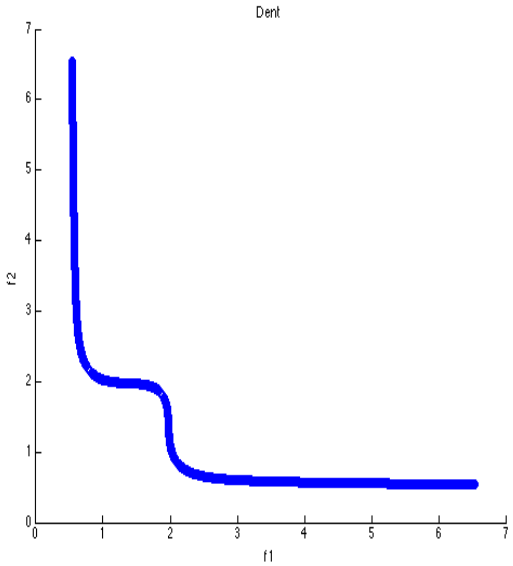


Figure B.3: Real Pareto front of the Dent function

B.4 ZDT1 Function

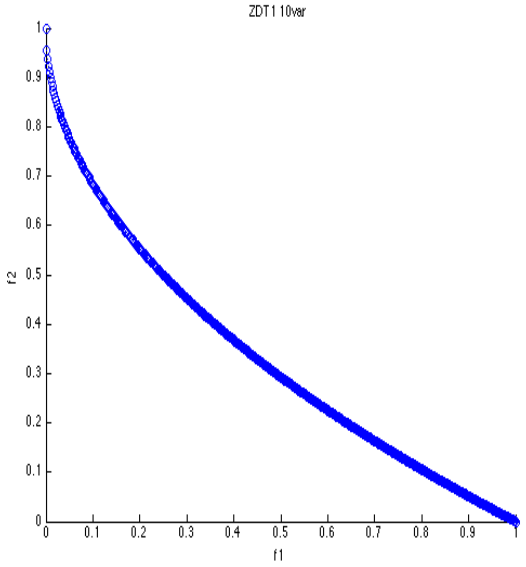


Figure B.4: Real Pareto front of the ZDT1 function

B.5 Binh tri-objective function

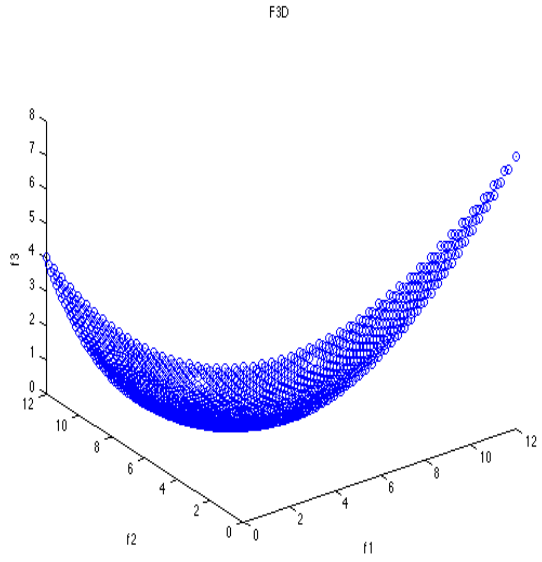


Figure B.5: Real Pareto front of the Binh tri-objective function

B.6 DTLZ1 Function

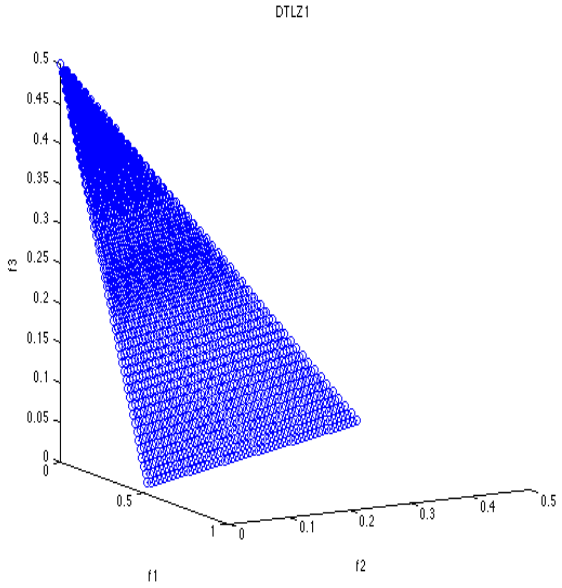


Figure B.6: Real Pareto front of the DTLZ1 function

B.7 DTLZ2 Function

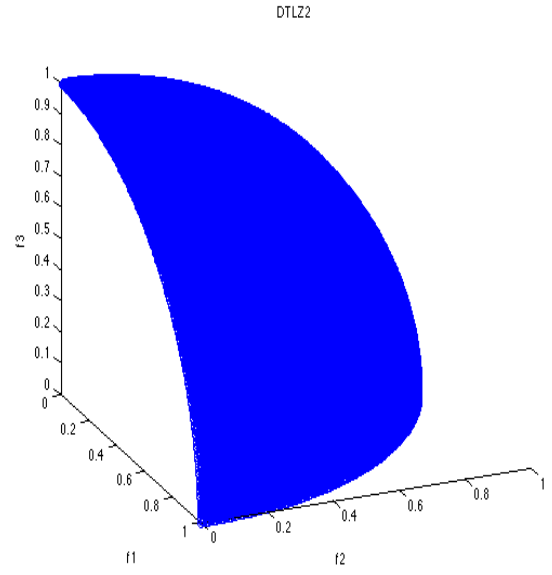


Figure B.7: Real Pareto front of the DTLZ2 function

B.8 Binh Integer Function

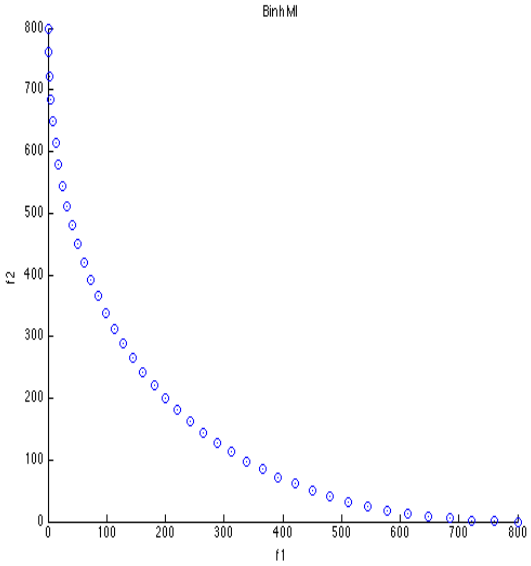


Figure B.8: Real Pareto front of the Binh Integer function

B.9 Dent Integer Function

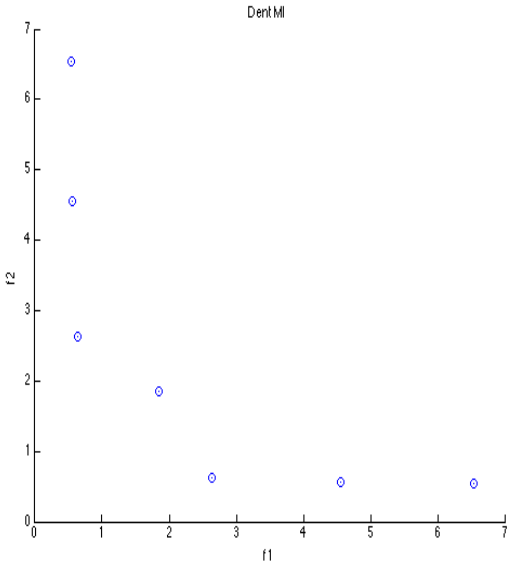


Figure B.9: Real Pareto front of the Dent Integer function

B.10 ZDT1 Integer Function

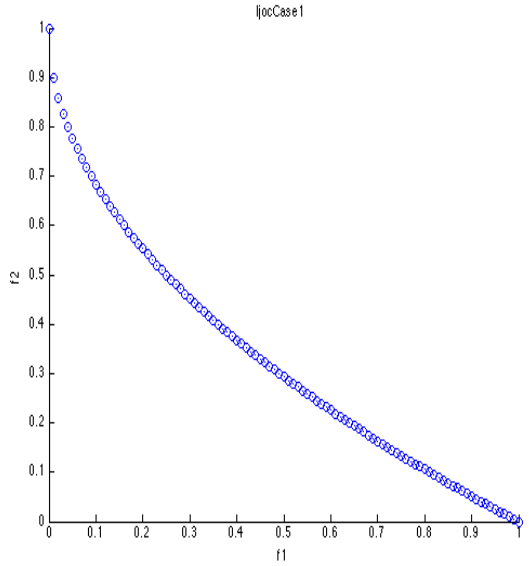


Figure B.10: Real Pareto front of the ZDT1 Integer function

B.11 ZDT2 Integer Function

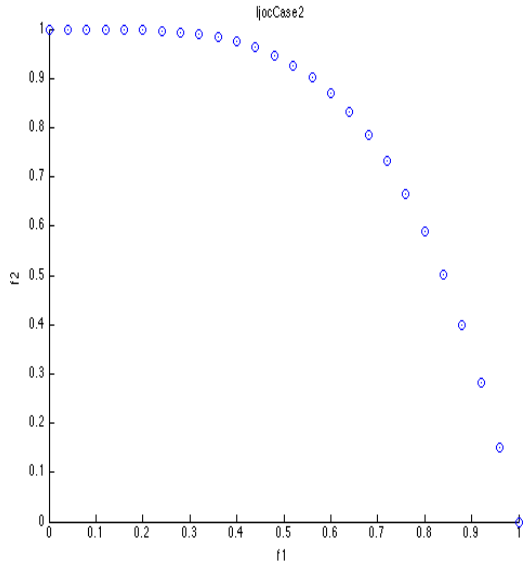


Figure B.11: Real Pareto front of the ZDT2 Integer function

B.12 Binh Mixed-Integer Function

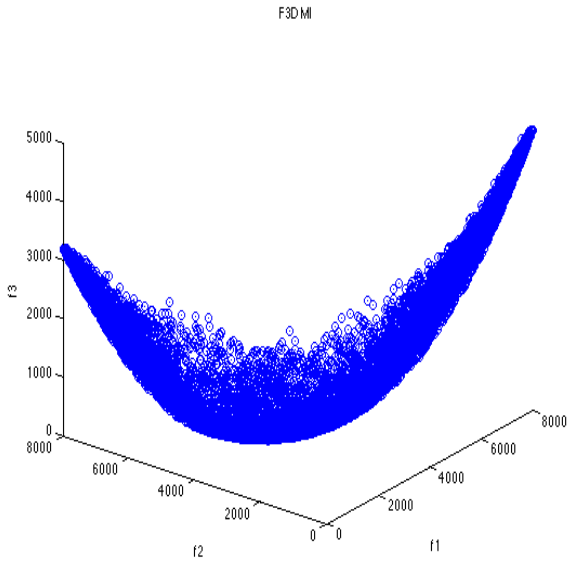


Figure B.12: Real Pareto front of the Binh3 MI function

Bibliography

[1] V. Pareto. *Manual of Political Economy*. A. M. Kelley, 1971.

[2] S. Ghodsypour and C. O'Brien. The Total Cost of Logistics in Supplier Selection under Conditions of Multiple Sourcing, Multiple Criteria and Capacity Constraints. *International Journal of Production Economics*, 73(1):15–27, 2001.

[3] B. Karpak, R. Kasuganti, and E. Kumzu. Multi-objective Decision Making in Supplier Selection: An Application of Visual Interactive Goal Programming. *The Journal of Applied Business Research*, 15(2):57–72, 1999.

[4] M. Ekhtiari and S. Poursafaryi. Multi-objective Stochastic Programming for Mixed Integer Vendor Selection Problem using Artificial Bee Colony Algorithm. *ISRN Artificial Intelligence*, 2013(1):1–13, 2013.

[5] W. Pan and F. Wang. A Multi-objective Model of Order Allocation under Considering Disruption Risk and Scenario Analysis in a Supply Chain Environment. In *International Conference on Global Economy, Commerce and Service Science*, volume 1, pages 325–327, Phuket, Thailand, 2014. Atlantis Press.

[6] P. McMullen and G. Frazier. Using Simulated Annealing to Solve a Multi-objective Assembly Line Balancing Problem with Parallel Workstations. *International Journal of Production Research*, 36(10):2717–2741, 1998.

[7] K. Miettinen. *Non Linear Multiobjective Optimization*. Springer, 1999.

[8] G. Eichfelder. *Addaptative Scalarization Methods in Multiobjective Optimization*. Springer, 2008.

[9] R. Marler and J. Arora. The Weighted Sum Method for Multi-objective Optimization: New Insights. *Structural and Multidisciplinary Optimization*, 41(6):853–862, 2010.

[10] J. Fliege. Gap Free Computation of Pareto Points by Quadratic Scalarizations. *Mathematical Methods of Operations Research*, 59(1):69–89, 2004.

[11] I. Das and J. Dennis. Normal Boundary Intersection: A New Method for Generating the Pareto Surface in Nonlinear Multicriteria Optimization Problems. *SIAM Journal on Optimization*, 8(3):631–657, 1998.

[12] A. Zhou et al. Multiobjective Evolutionary Algorithms: A Survey of the State of the Art. *Swarm and Evolutionary Computation*, 1(1):32–49, 2011.

[13] C. Coello, G. Lamont, and D. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, 2007.

[14] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley Interscience Series in Systems and Optimization. John Wiley & Sons, 2001.

[15] Oliver Schütze, Marco Laumanns, Emilia Tantar, Carlos A. Coello Coello, and El-Ghazali Talbi. Computing Gap Free Pareto Front Approximations with Stochastic Search Algorithms. *ACM Journal in Evolutionary Computation*, 18(1):65–96, 2010.

[16] M. Dellnitz, O. Schütze, and T. Hestermeyer. Covering Pareto Sets by Multilevel Subdivision Techniques. *Journal of Optimization Theory and Applications*, 124(1):113–136, 2005.

[17] J. Jahn. Multiobjective Search Algorithm with Subdivision Techniques. *Computational Optimization and Applications*, 35(2):161–175, 2006.

[18] C. Hernandez, Y. Naranjani, Y. Sardahi, W. Liang, O. Schütze, and J. Q. Sun. Simple Cell Mapping Method for Multi-objective Optimal Feedback Control Design. *International Journal of Dynamics and Control*, 1(3):231–238, 2013.

[19] C. Hernandez, O. Scütze, and J. Q. Sun. Computing the Set of Approximate Solutions of a Multi-objective Optimization Problem by Means of Cell Mapping Techniques. In *EVOLVE - A Bridge between Probability, Set Oriented Numerics and Evolutionary Computation*, volume 4 of *Advances in Intelligent Systems and Computing*, pages 171–188. Springer, 2013.

[20] J. Fliege, L. Drummond, and B. Svaiter. Newton's Method for Multi-objective Optimization. *SIAM Journal on Optimization*, 20(2):602–626, 2009.

[21] Z Povalej. Quasi-Newton's Method for Multi-objective Optimization. *Journal of Computational and Applied Mathematics*, 255(1):765–777, 2014.

[22] O. Schütze, A. Lara, and C. Coello. The Directed Search Method for Multi-objective Optimization Problems. In *EVOLVE - A Bridge between Probability, Set Oriented Numerics and Evolutionary Computation*, volume 2 of *Studies in Computational Intelligence*, pages 153–168. Springer, 2013.

[23] J. Rakowska, T. Haftka, and L. Watson. Multi-objective Control-Structure Optimization via Homotopy Methods. *SIAM Journal on Optimization*, 3(3):654–667, 1993.

[24] B. Martin, A. Goldsztejn, L. Granvilliers, and C. Jermann. On Continuation Methods for Non-linear Bi-objective Optimization: Towards a Certified Interval-Based Approach. *Journal of Global Optimization*, pages 1–14, 2014.

[25] A. Potschka, F. Logist, J. Van Impe, and H. Bock. Tracing the Pareto Frontier in Bi-objective Optimization Problems by ODE Techniques. *Numerical Algorithms*, 57(2):217–233, 2011.

[26] H. Wang. Zigzag Search for Continous Multi-objective Optimization. *INFORMS Journal on Computing*, 25(4):654–665, 2013.

[27] C. Hillermeier. *Nonlinear Multiobjective Optimization: A Generalized Homotopy*. Springer, 2001.

[28] L. Loomis and S Sternberg. *Advanced Calculus*. World Scientific, 2014.

[29] J. Onwunalu and L. Durlofsky. Application of a Particle Swarm Optimization Algorithm for Determining Optimum Well Location and Type. *Computational GeoScience*, 6(1):183–198, 2010.

[30] H. Wang, D. Echeverria, L. Durlofsky, and A. Cominelli. Optimal Well Placement under Uncertainty using a Retrospective Optimization Framework. *Society of Petroleum Engineers Journal*, 17(1):112–121, 2012.

[31] R. Gutierrez, G. Valencia, O. Rodríguez, and L. Trujillo. Systematic Selection of Tuning Parameters for Efficient Predictive Controllers using a Multi-objective Evolutionary Algorithm. Technical report, ITT, Tijuana, Mexico, 2014.

[32] E. Bjornson and E. Jorswieck. Optimal Resource Allocation in Coordinated Multi-cell Systems. *Foundations and Trends in Communications and Information Theory*, 9(3):113–381, 2013.

[33] O. Schütze, X. Esquivel, A. Lara, and C. Coello. Using the Averaged Hausdorff Distance as a Performance Measure in Evolutionary Multiobjective Optimization. *IEEE Transactions on Evolutionary Computation*, 16(4):504–522, 2012.

[34] O. Schütze, S. Alvarado, C. Segura, and R. Landa. Gradient Subspace Approximation: A Directed Search Method for Memetic Computing. Technical report, Computer Science Department, CINVESTAV-IPN, Mexico D.F., 2014.

[35] O. Schütze, A. Dell'Aere, and M. Dellnitz. On Continuation Methods for the Numerical Treatment of Multi-objective Optimization Problems. In *Practical Approaches to Multi-Objective Optimization*, Internationales Begegnungs und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.

[36] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, 2006.

[37] C. Coello. Evolutionary Multiobjective Optimization: A Historical View of the Field. *IEEE Computational Intelligence Magazine*, 1(1):28–36, 2006.

[38] H. Kuhn and A. Tucker. Nonlinear Programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492, Berkeley, California, 1951. University of California Press.

[39] E. Algower and K. Georg. *Introduction to Numerical Continuation Methods*. Classics in Applied Mathematics. SIAM, 2003.

[40] A. Björck. *Numerical Methods for Least Squares Problems*. SIAM, 1996.

[41] H. Wang. Zigzag Search for Discrete Multi-objective Optimization. Technical report, Industrial and Systems Engineering, Busch Campus, Rutgers University, N. J., USA, 2014.

[42] R. Hooke and T. Jeeves. Direct Search Solution of Numerical and Statistical Problems. *Journal of the Association for Computing Machinery (ACM)*, 8(2):212–229, 1961.

[43] A. Engelbrecht. *Computational Intelligence, an Introduction*. WILEY, 2007.

[44] R. Eberhart and Y. Shi. *Computational Intelligence, Concepts to Implementations*. Morgan Kaufmann, 2007.

[45] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[46] Q. Zhang and H. Li. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

[47] D. Van Veldhuizen. *Multi-objective Evolutionary Algorithms: Classifications, Analysis and New Innovations*. PhD thesis, Department of Electrical and Computer Engineering. Graduate School of Engineering. Air Force Institute of Technology, Wright-Patterson AFB, Ohio, May, 1999.

[48] C. Coello and N. Cruz. Solving Multi-objective Optimization Problems using an Artificial Immune System. *Genetic Programming and Evolvable Machines*, 6(2):163–190, 2005.

[49] D. Huttenlocher, G. Klanderman, and W. Rucklidge. Comparing Images using the Hausdorff Distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, 1993.

[50] O. Schütze and A. Martín. Directed Search 2. Technical report, Computer Science Department at CINVESTAV Zacatenco, Mexico, 2014.

[51] fmincon Function Description. http://www.mathworks.com/help/optim/ug/fmincon.html#bribv6f. Accessed: November, 2015.

[52] S. Schäffler, R. Schultz, and K. Weinzierl. A Stochastic Method for the Solution of Unconstrained Vector Optimization Problems. *Journal of Optimization: Theory and Applications*, 114(1):209–222, 2002.

[53] A. Martín. Pareto Tracer: A Predictor Corrector Method for Multi-objective Optmization Problems. Master's thesis, Computer Science Department at CINVESTAV Zacatenco, 2014.

[54] NGPM: A NSGA-II Program in Matlab. http://www.mathworks.com/matlabcentral/fileexchange/31166-ngpm-a-nsga-ii-program-in-matlab-v1-4. Accessed: November, 2015.

[55] L. Song. NGPM: A NSGA-II Program in Matlab. Technical report, College of Astronautics at Northwestern Polytechnical University, China, 2011.

[56] K. Witting. *Numerical Algorithms for the Treatment of Parametric Multiobjective Optimization Problems and Applications*. PhD thesis, Universität Paderborn, Febraury, 2012.

[57] E. Zitzler, K. Deb, and L Thiele. Comparison of Multi-objective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195, 2000.