Centro de Investigación y de Estudios Avanzados

del Instituto Politécnico Nacional

UNIDAD ZACATENCO

DEPARTAMENTO DE COMPUTACIÓN

Mapeo de celda a celda para problemas de optimización de dos niveles.

Tesis que presenta

Jesús Fernández Cruz

para obtener el Grado de

Maestro en Ciencias en Computación

Director de tesis:

Dr. Oliver Steffen Schütze

México, DF **Noviembre del 2014**

Centro de Investigación y de Estudios Avanzados

del Instituto Politécnico Nacional

ZACATENCO Campus

Computer Science Departmen

Cell-to-Cell Mapping for Bilevel Optimization Problems

Submitted by

Jesús Fernández Cruz

As fulfillment of the requirement for the degree of

Ms. C. Computer Science

Advisor:

Dr. Oliver Steffen Schütze

México, DF **November 2014**

Resumen

Cada día es más frecuente que encontremos problemas de optimización de dos niveles simultáneos (POB). Cada nivel típicamente es un problema de optimización multi-objetivo (POM); dónde estas dos funciones pueden ser cooperativas o no.

En esta tesis, presentamos métodos paralelos orientados a conjuntos para resolver esta clase de problemas. En particular, nos enfocamos al problema de encontrar el conjunto de soluciones óptimas de Pareto, pero también para el caso de los POMs presentamos métodos paralelos para encontrar el conjunto de soluciones aproximadas. También se presentan métodos paralelos para hallar la familia de frentes de Pareto de problemas de optimización paramétricos (POP). Finalmente, se presenta un estudio en el cual se analiza el paralelismo teórico y práctico que tienen estos métodos.

Abstract

Nowadays, it is more common to deal with bilevel optimization problems (BOP); where typically each level is a multi-objective optimization problem (MOP). These two objective functions can be cooperative, or not.

In this thesis, we present parallel set oriented methods for the treatment of these problems. In particular, we address the problem of computing the set of optimal solutions; also, for the case of MOPs, we present parallel methods for finding the set of approximate solutions. Parallel methods are also presented to find the family of Pareto fronts of a given parametric optimization problem (POP). Finally, a study that analyzes the theoretical and practical parallelism, of this methods, is presented.

Agradecimientos

Quisiera agradecer a mi padre y mi madre, por todo el apoyo incondicional que me dieron, así como también la educación que me impartieron.

También quisiera agradecer a mis hermanas por su apoyo, ya que sin él no hubiera podido alcanzar mis metas.

A mi esposa Rocío por acompañarme estos dos años, donde tuvo que demostrar paciencia y amor al no poder siempre estar con ella.

A mis amigos Marco, RAL y Pesca que me ayudaron a lo largo de mi vida a demostrar que es muy divertida.

Agradezco al CONACyT por la beca que me dio para poder realizar mis estudios de maestría.

También agradezco al CINVESTAV, en particular al departamento de computación, a todos mis profesores, secretarias y compañeros, ya que con ellos compartí experiencias y conocimiento importante, para lograr de mí una mejor persona.

Finalmente quisiera agradecer a mi asesor ya que siempre tuve de él, apoyo, consejos y sobre todo paciencia, para que yo sea un mejor académico.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Bilevel programming problems (BLPs) represent a special class of optimization problems with two levels of optimization that appear e.g. in several practical problem solving tasks. BLPs are different from the common optimization problems, as they contain a nested optimization task within the constraints of another optimization problem. The nested structure of the overall problem requires that a solution to the upper level problem is feasible if and only if it is an optimal solution to the lower level problem.

1.1 Motivation

Bilevel problems have been widely studied by researchers, in the field of classical as well as evolutionary optimization. But in the context of multi-objective bilevel optimization problems, which contain multiple objectives at one or both levels, few recent studies in classical and evolutionary optimization have been done.

Recently, it has been discovered that these problem formulations can be very useful in automatic control applications. For example, they can be used to compute the worst case deviation of a suboptimal model predictive control scheme compared to the optimal one [7].

1.2 The Problem

Bilevel optimization constitutes a challenging class of optimization problems due to its high dimensionality, the computational cost to obtain a feasible set for the upper level task, and they are in most cases non-convex, therefore hard to solve for some global search methods.

The greatest challenge in handling bilevel optimization problems is based on the fact that unless a solution is optimal for the lower level problem, it cannot be feasible for the overall problem. This requirement, in some sense, disallows any approximate optimization algorithm (including Evolutionary Algorithms) to be used for solving the lower level task.

The simple cell mapping method [18] gives us a useful tool to obtain the attractors and basins of attractors of a dynamical system. This approach proposes to discretize by dividing the state space in bigger hypercubes called cells. The evaluation of the dynamical system is then reduced to a new function, which is defined not in R^n, but on the cell space. In this case we restrict ourselves to functions that are strictly deterministically defined. Thus, we can extend this idea to the context of bilevel optimization.

1.3 Objectives

1.3.1 General Objective

To develop a method that uses simple cell mapping for the treatment of single-objective and multi-objective bilevel optimization problems.

1.3.2 Particular Objectives

- To parallelize the Simple Cell Mapping method (pSCM).
- To develop a method that solves single-objective bilevel optimization problems

using pSCM.

- To develop a method that solves multi-objective bilevel optimization problems using pSCM.
- To solve a real world problem with the resulting method.

1.4 Contributions

- Set oriented parallel algorithms for global multi-objective optimization problems
  - parallel version of the SCM for multi-objective optimization.
  - pSCM for multi-objective optimization problems.
  - pSCM for the computation of the set of approximate solutions
- Set oriented parallel algorithm for global parametric optimization problems
  - pSCM for multi-objective parametric optimization problems.
- Set oriented parallel algorithms for global bilevel optimization problems
  - pSCM for single-objective bilevel optimization problems.
  - pSCM for multi-objective bilevel optimization problems.
- Collaboration with the University of California, USA.
- Collaboration with the University of Tianjin, China.
- The thesis resulted in the following publications
  - Contribution at EVOLVE 2014 international conference [11].
  - Contribution at EVOLVE 2014 international conference [12].
  - Contribution at EVOLVE 2014 international conference [25].
  - Contribution at Journal of Sound and Vibration [27].

1.5 Organization of the thesis

This thesis consists of seven chapters, including this introductory chapter. The remainder of this document is organized as follows:

Chapter 2 presents the basic concepts of multi-objective optimization, parametric optimization, bilevel optimization and dynamical systems that are fundamental for understanding the current work. Further, we review some of the methods for solving a bilevel optimization problem along with some methods for global analysis of dynamical systems.

Chapter 3 is devoted to present the parallel simple cell mapping method in the context of multi-objective optimization. In this chapter, we discuss the key elements to adapt this method and further on present numerical results on some academic models, also, the chapter describe the simple cell mapping for the set of approximate solutions where we describe the elements that are incorporated to the method to compute this set.

Chapter 4 describes pSCM for multi-objective parametric optimization. In this chapter we present an algorithm for the treatment of this kind of problems, along with some numerical results on some academic models.

Chapter 5 includes the algorithm for the treatment of bilevel optimization problems, this chapter also includes some numerical results and comparisons with other method.

Chapter 6 presents a real world application solved with pSCM and the interpretation of the numerical results. Finally, Chapter 7 contains the conclusions and some possible future ideas to be developed from this work.

Chapter 2

Background and Related Work

2.1 Optimization

Optimization is the selection of the best element(s) with regard to some criteria, from a set of feasible alternatives.

In this section we review the theoretical background on multi-objective optimization and a definition of optimality.Then we describe the general background on bilevel optimization (single-objective and multi-objective). Finally,we present some notion of optimality for bilevel optimization problems.

2.1.1 Multi-objective Optimization

Multi-objective optimization is a kind of optimization that involves more than one objective function to be optimized simultaneously. Multi-objective optimization has been applied in many fields of science, including engineering, economics and logistics where optimal decisions need to be taken in the presence of trade-offs between two or more conflicting objectives. Two examples are minimizing cost while maximizing comfort when buying a car, and maximizing performance whilst minimizing fuel consumption of a vehicle.

A general multi-objective optimization problem (MOP) can be stated as follows:

$$\begin{aligned}
&\underset{x\in\mathbb{R}^n}{\text{minimize}}\;\; F(x) = [f_1(x), ..., f_k(x)]^T \\
&\text{subject to} \\
&g_i(x) \leq 0 \quad i = 1, \ldots, l, \\
&h_j(x) = 0 \quad j = 1, \ldots, m.
\end{aligned} \tag{2.1}$$

Let $\Omega \subset \mathbb{R}^n$ be a feasible region defined by

$$\Omega = \{x \in \mathbb{R}^n | g(x) \leq 0 \text{ and } h(x) = 0\}, \tag{2.2}$$

where $F : \Omega \to \mathbb{R}^k$ is a vector consisting of the objective functions

$$f_i : \Omega \to \mathbb{R}, i = 1, \ldots, k, \tag{2.3}$$

$x \in \Omega$ is known as a *decision vector*, $g_i : \mathbb{R}^n \to \mathbb{R}, i = 1, \ldots, l$ is an inequality constraint and $h_i : \mathbb{R}^n \to \mathbb{R}, i = 1, \ldots, m$ is an equality constraint. In case there are no constraints the MOP is known as unconstrained. We can also see that in case $k = 1$ the problem is a single-objective optimization problem (SOP). It is important to notice that we could also state the MOP as a maximization problem, however, any maximization problem can be stated as a minimization problem, by multiplying the objective function vector by -1.

2.1.2 Pareto Optimality

Once we know the problem, we must define what we seek, as in the example of the choice of the car described in the Section 2.1.1. There is a set of feasible cars, but among all of them, it is necessary to choose the best with certain characteristics of speed and fuel consumption, but, what happens when we do not know which one is the best? Or when our job is not to choose? For this task we can use Pareto dominance relation and choose a set of non-dominated decision vectors among the entire search space.

Definition 2.1.1 Pareto dominance

1. *Let $y, x \in \mathbb{R}^n$, then x is less than y $(x <_p y)$ if $x_i < y_i$ for all $i = 1, \ldots, n$. The relation $\leq_p$ is defined analogously.*

2. *A vector $y \in \Omega$ is called dominated by a vector $x \in \Omega$ $(x \prec y)$ if $F(x) \leq_p F(y)$* and *$F(x) \neq F(y)$; else, it is called non-dominated by x.*

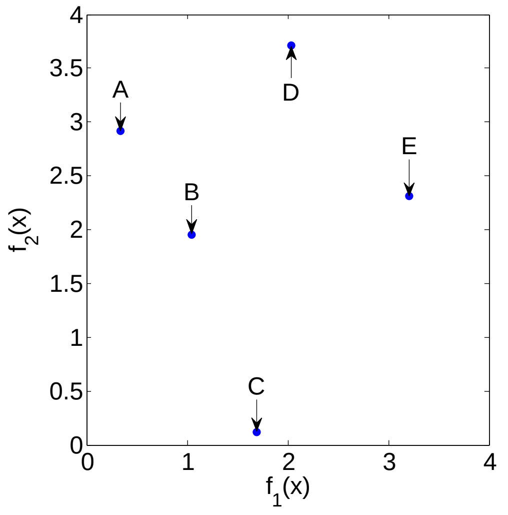


Figure 2.1: Example of Pareto dominance, where the point A dominates the point E, and the points B and C dominates the points E and D.

Figure 2.1 shows an example of Pareto dominance, which is a way to define which point is 'better' according to another point.

Definition 2.1.2 Pareto optimal solution

A point $x \in \Omega$ is called Pareto point if there is not $y \in \Omega$ such that $y \prec x$.

Usually, in the MOPs, there exists a set of Pareto optimal solutions, this set is called Pareto set, and its image is called Pareto front.

Definition 2.1.3 *1. The set of all Pareto optimal solutions is called the Pareto set, i.e. $P = \{x \in \Omega : x$ is a Pareto optimal point $\}$.*

2. The image of P, $F(P)$, is called the Pareto front.

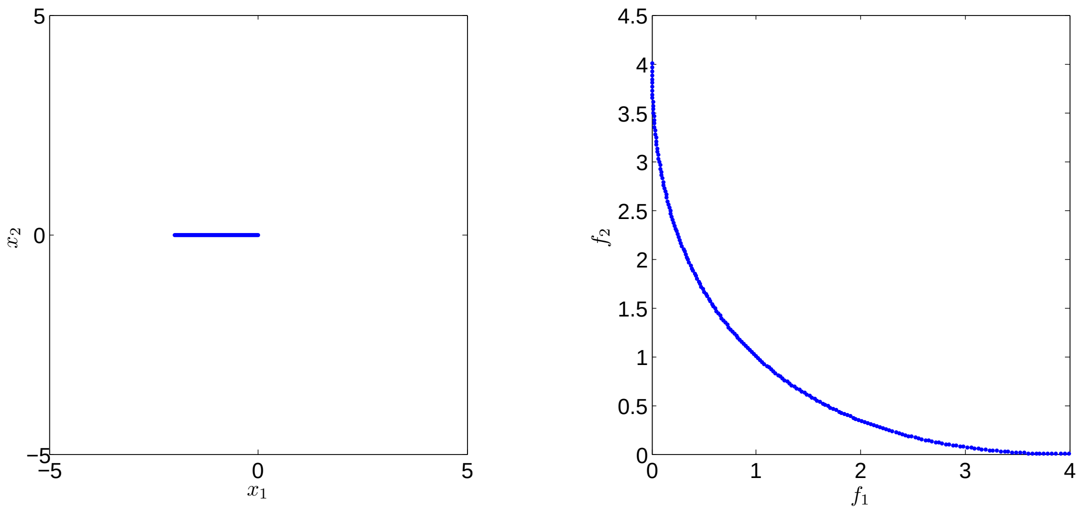


Figure 2.2: Pareto set (left) and Pareto front (right) of Leumman's problem

Figure 2.2 shows the Pareto set of Leumman's problem (see problem 2.4) [22], which is a line where $x \in [-2, 0]$ and $y = 0$, and the image of the Pareto set, typically is an k-dimensional object.

$$\begin{aligned} &\underset{x \in \mathbb{R}^2}{\text{minimize}} \;\; F(x) = [f_1(x), f_2(x)]^T \\ &f_1(x) = x_1^2 + x_2^2 \\ &f_2(x) = (x_1 + 2)^2 + x_2^2. \end{aligned} \tag{2.4}$$

2.1.3 Parameter dependant Multi-objective Optimization

When we are dealing with optimization in real world applications, we often find some variables that cannot be optimized, i.e. those variables no longer belong to the design variables. Instead, those variables are external. For example, a car's computer, when is processing the best response for a given curve, we can optimized the velocity, the angle of each tire, even the oil pressure; but, the wind is a parameter that cannot be optimize. Therefore, a new solution is obtained given a different wind parameter.

A parameter dependent optimization problem, given a parameter $\lambda \in \mathbb{R}^p$ can be written as:

$$\begin{aligned}
&\underset{x \in \mathbb{R}}{\text{minimize}} \;\; F(x,\lambda) = [f_1(x,\lambda), ..., f_k(x,\lambda)]^T \\
&\text{subject to} \\
&g_i(x,\lambda) \leq 0 \quad i = 1, \ldots, l, \\
&h_j(x,\lambda) = 0 \quad j = 1, \ldots, m,
\end{aligned} \tag{2.5}$$

where: $F : \mathbb{R}^n \times [\lambda_l, \lambda_u] \rightarrow \mathbb{R}^k$. $\lambda_h \in [\lambda_{hl}, \lambda_{hu}]$, λ_{hl} and λ_{hu} are the lower bounds and upper bounds respectively. let Ω be the feasible set; every element in Ω complies with the equality and inequality constraints. The feasible *decision vectors* are those $x \in \Omega \subseteq \mathbb{R}^n$.

As a result of this kind of optimization, we obtain a particular Pareto front and Pareto set for each λ value, i.e., the original function F is changing for each λ value.

2.1.4 Bilevel Optimization

Bilevel optimization problems (BOPs) are a special class of optimization problems with two levels of optimization that appear e.g. in several practical problem solving tasks. BOPs are different from the common optimization problems, as they contain a nested optimization task within the constraints of another optimization problem. The nested structure of the overall problem requires that a solution to the upper level problem is feasible if and only if it is an optimal solution to the lower level problem.

A general single objective bilevel optimization problem (BSOP) can be formulated as follows[1]:

$$
\begin{aligned}
&\underset{x_u \in \Omega_u, x_l \in \Omega_l}{\text{minimize}} \, f_u(x_u, x_l)\\
&\text{subject to}\\
&G_i(x_u, x_l) \leq 0 \quad i = 1, \ldots, L,\\
&H_j(x_u, x_l) = 0 \quad j = 1, \ldots, M,\\
&x_l \in \underset{l \in L}{\text{argmin}} \left\{ \begin{array}{l} f_l(x_u, x_l) \\ g_i(x_u, x_l) \leq 0, i = 1, \ldots, l, \\ h_j(x_u, x_l) = 0, j = 1, \ldots, m \end{array} \right\},
\end{aligned}
\tag{2.6}
$$

where f_u and f_l represent the *objective functions* at the upper and lower level respectively, x_u represents the upper level *decision vector* and x_l represents the lower level *decision vector*. Inequality constraints at the upper and lower levels are represented by G_i and g_i, respectively, analog for equality constraints H_j and h_j. Ω_u and Ω_l are the bound constraints for the upper level *decision vector* and lower level *decision vector*, respectively.

In the above formulation, a vector $x^{'} = (x^{'}_u, x^{'}_l)$ is considered to be feasible at the upper level, if it satisfies all the upper level constraints and $x^{'}_l$ is optimal at the lower level for a given $x^{'}_u$. We observe in this formulation that the lower level problem is a parametrized constraint to the upper level.

We can restate the single-objective bilevel program (Equation (2.6)) as a multi-objective bilevel program (BMOP).

$$\begin{aligned}
&\min_{x_u\in\Omega_u, x_l\in\Omega_l}. F_u(x_u,x_l) = [f'_{u1}(x_u,x_l),...,f'_{uk}(x_u,x_l)]^T \\
&\text{subject to} \\
&G_i(x_u,x_l) \leq 0 \quad i = 1,\dots,L, \\
&H_j(x_u,x_l) = 0 \quad j = 1,\dots,M, \\
&x_l \in \underset{l\in L}{\text{argmin}} \left\{ \begin{array}{l} F_l(x_u,x_l) = [f'_{l1}(x_u,x_l),...,f'_{ln}(x_u,x_l)]^T : \\ g_i(x_u,x_l) \leq 0, i = 1,\dots,l, \\ h_j(x_u,x_l) = 0, j = 1,\dots,m \end{array} \right\}.
\end{aligned} \tag{2.7}$$

Hereby, $F_u(u,l)$ and $F_l(u,l)$ represent the vectors of objective functions where $f'_{ui}, f'_{lj} : \mathbb{R}^n \rightarrow \mathbb{R}$, $i = 1,\dots,k$, $j = 1,\dots,m$, are the objectives, x_u represents the *upper level decision vector* and x_l represents the *lower level decision vector.* G_i, g_i, H_j and h_j denote the inequality and equality constraints for the upper and lower level, respectively. A vector $x' = (x'_u, x'_l)$ is considered feasible at the upper level, if it satisfies all the upper level constraints and x'_l is optimal at the lower level for the given x'_u. We observe in this formulation that the lower level problem is a parametrized constraint to the upper level.

A football match is an example of a BMOP, since a player is the lower level optimization task, and the player's team is the upper level optimization task. The best of the team is based on the strategy (upper level decision variables) that has the best plays (lower level decision variables) of each player, and the best strategy among all possibilities is the one that give to that team the maximum advantage over the opponent's team.

2.1.5 Pareto Optimality

Since the BMOPs are a special class of MOPs, we can use Pareto optimality as a way to select the 'best' solutions, but it is necessary to perform two different tests. A test is performed on the lower level objective functions, which is needed to get a feasible set for the upper optimization task; a second dominance check is computed on the upper level objective functions in order to obtain our final Pareto front and Pareto set.

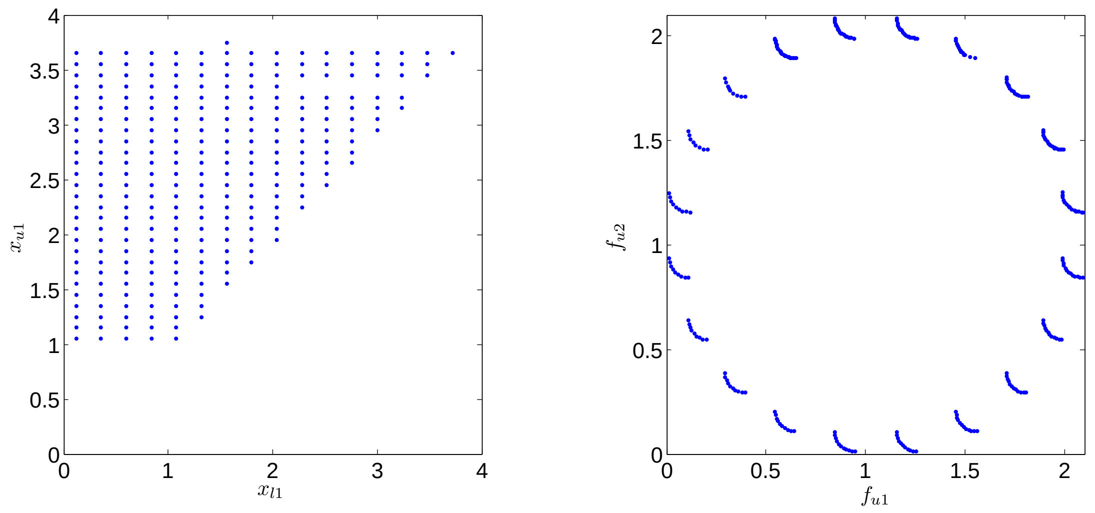


Figure 2.3: The feasible set (left) and its image (right) of DS1 problem.

Figure 2.3 shows the feasible set for the upper optimization task of DS1 problem (see problem 2.8) and Figure 2.4 presents its final result after a dominance test over the upper level objective function.

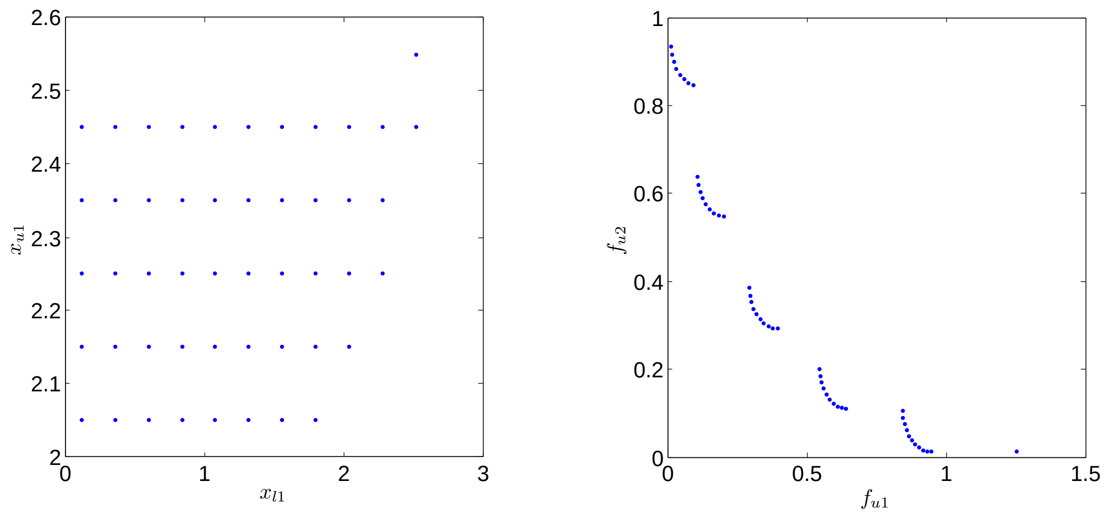


Figure 2.4: Pareto set (left) and Pareto front (right) of DS1 problem.

$$
\begin{aligned}
\min_{x_u \in \Omega_u, x_l \in \Omega_l}. f_{u1}(x_u, x_l) =& (1 + r - \cos(\alpha \pi x_{l1})) + \sum_{i=2}^{K} (x_{li} - (i-1)/2)^2 \\
&+ r \sum_{i=2}^{K} (x_{ui} - x_{li})^2 - r \cos((\pi x_{u1})/(2x_{l1})) \\
f_{u2}(x_u, x_l) =& (1 + r - \sin(\alpha \pi x_{l1})) + \sum_{i=2}^{K} (x_{li} - (i-1)/(2))^2 \\
&+ r \sum_{i=2}^{K} (x_{ui} - x_{li})^2 - r \sin((\pi x_{u1})/(2x_{li}))
\end{aligned}
$$

subject to

$$
x_l \in \underset{l \in L}{\operatorname{argmin}} \left\{
\begin{array}{l}
f_{l1}(x_u, x_l) = x_{u1}^2 + \sum_{i=2}^{K} (x_{ui} - x_{li})^2 + \sum_{i=2}^{K} 10(1 - \cos(4\pi(x_{ui} - x_{li}))) \\
f_{l2}(x_u, x_l) = \sum_{i=1}^{K} (x_{ui} - x_{li})^2 + \sum_{i=2}^{K} 10|(1 - \cos(4\pi(x_{ui} - x_{li})))| \\
-K <= x_i <= K, \ 1 <= y_1 <= 4, \ -K <= y_j <= K \\
\text{for} \quad i = 1, \ldots, K, j = 1, \ldots, K.
\end{array}
\right\}. \tag{2.8}
$$

2.2 Parallel Computing

Amdahl's law [29] indicates the maximum possible speedup of a program as a result of parallelization.

The maximum acceleration, which can be obtained given n processors, is represented by:

$$S(n) = \frac{1}{p_s + \frac{1}{n}(1 - p_s)}, \tag{2.9}$$

where $p_s \in [0, 1]$ is the sequential part of the program, i.e., the part that cannot be parallelized.

Amdhal's law also states that there exists an upper bound for the acceleration regardless of the number of processing elements available.

The maximum theoretical acceleration (upper bound) is given in the limit where the number of processors approaches infinity:

$$\lim_{n \to +\infty} S(n) = \lim_{n \to +\infty} \frac{1}{p_s + \frac{1}{n}(1 - p_s)} = \frac{1}{p_s}, \tag{2.10}$$

where $p_s \in [0, 1]$ is the sequential part of the program, i.e. the part that cannot be parallelized.

Gustafson's law [29] says that computations involving arbitrarily large data sets can be efficiently parallelized. Gustafson's law provides a counterpoint to Amdahl's law fixed data set size.

According to Gustafson's law, the maximum acceleration which can be obtained given n processors, is represented by:

$$S(n) = n - p_s(n - 1), \tag{2.11}$$

where $p_s \in [0, 1]$ is the sequential part of the program, i.e., the part that cannot be parallelized.

Gustafson's law addresses the shortcomings of Amdahl's law, which does not fully exploit the computing power that becomes available as the number of machines increases. Gustafson's law instead proposes that programmers tend to set the size of problems to use the available equipment to solve problems within a practical fixed time. Therefore, if faster (more parallel) equipment is available, larger problems can be solved in the same time.

2.3 Cell Mapping Techniques

Cell mapping techniques were originally designed for the global analysis of general dynamical systems ([18, 6]). Algorithms of that type discretize the entire search space into cells and consider instead of the point-wise iteration of the dynamical system F, the dynamics on the cells induced by F.

A dynamical system [18] is a system whose state changes with time (t). Two main types of dynamical systems are encountered:

- Those for which the time variable is discrete ($t \in \mathbb{N}$), called maps.
- Those for which the time variable is continuous ($t \in \mathbb{R}$), called flows.

Discrete dynamical systems can be presented as the iteration of a function

$$x_{t+1} = f(x_t), \tag{2.12}$$

where x_t is the actual iteration, $F : \mathbb{R}^n \to \mathbb{R}^n$ is the motion or dynamics of the system and x_{t+1} is the next iterate.

When studying the global behavior of a nonlinear dynamic system, one usually focuses on the following steps:

- Formulation of the equations of motion.
- Localization of stationary, periodic, quasiperiodic and chaotic solutions of the system.
- Determination of the stability properties of the solutions.

- Study of changes in the solutions (in forms, magnitude, number, stability and type) when varying system parameters.

In nonlinear systems, one often deals with the coexistence of various stable solutions, also called attractors. For each attractor, a subset of the state space exists, containing all initial values leading to that attractor, the so-called basin of attraction. Which attractor the system will lead to depends on the initial conditions of the system.

When using numerical techniques to solve problems, roundoff errors are introduced due to the computer's limited precision. Moreover, in experimental methods, a limit of measurement accuracy exists, which means that in both numerical and experimental methods, desired quantities cannot be obtained in an exact manner.

According to Hsu[18], this implies that a state variable, describing a part of the state of a dynamical system cannot be regarded as a continuum, which can take every possible value $x_i \in \mathbb{R}$. Instead it is better to consider as a discrete quantity, and its range as a collection of n-dimensional intervals, where n is the state space dimension. Such discretized space is called a cell state space.

Consider a dynamical system with Euclidian state space $\mathbb{R}^n$, $n \in \mathbb{N}, n \geq 1$. Generally, the state of a dynamical system will be restricted to a bounded subset of the state space. For convenience, we consider this subset, denoted by Ω, to be rectangular. Let $x = (x_1, \ldots, x_n)$ be the state vector, then for each state variable x_i a lower and upper boundaries l_i and u_i exists. Hence:

$$l_i \leq x_i \leq u_i, i = 1, \ldots, n. \tag{2.13}$$

Having defined Ω, we divide it into cells. Cells can be of an arbitrary form, as long as they fill up Ω. The division of Ω in rectangular cells can be done by dividing each interval $[l_i, u_i]$ into M_i intervals of equal length h_i where:

$$h_i = \frac{u_i - l_i}{M_i}, i = 1, \ldots, n. \tag{2.14}$$

In this way, Ω has been divided into M rectangular cells as shown in Figure 2.5, with:

$$M = \prod_{i=1}^{n} M_i. \tag{2.15}$$

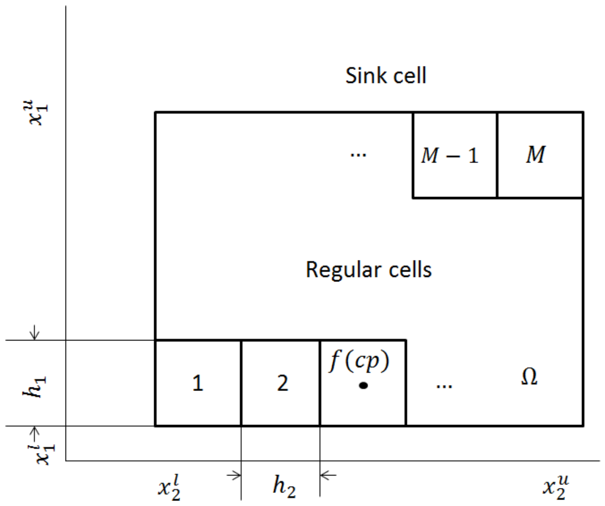


Figure 2.5: The division of Ω into M rectangular cells.

Each cell is denoted by an index $j \in 1, \ldots, M$. The region $\mathbb{R}^n \setminus \Omega$ is called the sink cell and is denoted by index 0. The complexity given by Equation 2.15 indicates that this method works only for low dimensional problems. All possible states within one cell are denoted by the same index and are treated as one.

The mapping $C : \mathbb{N} \to \mathbb{N}$ is called a simple cell mapping (SCM), by $\xi(n+1) = C(\xi(n))$ is implied that the next state of the system is completely determined by its current state as shown in Figure 2.6.

When the system enters the sink cell, its evolution is no longer followed. We define $C(0) = 0$. Under SCM, two kinds of regular cells are obtained: periodic and transient cells.

A cell ξ that satisfies $\xi = C^m(\xi)$ for some $m \in \mathbb{N}$, is called a periodic cell with period m, or $P - m$ cell, here C^m denotes the cell mapping C applied m times, all the cells $C^{m-1}(\xi)$, $C^{m-2}(\xi)$, $\ldots$ are also $P - m$ cells. By definition the sink cell is a

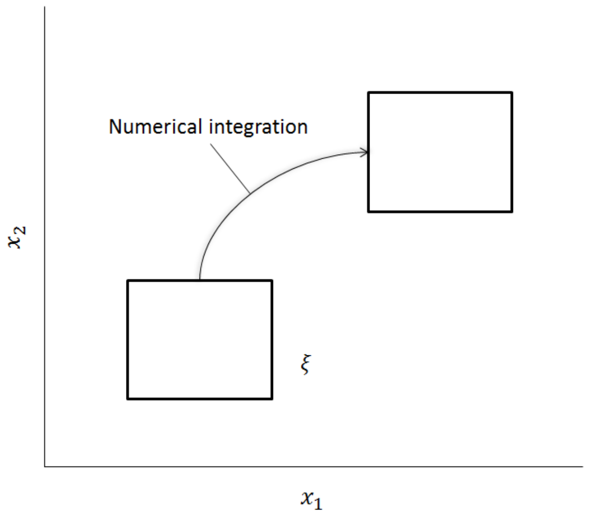


Figure 2.6: The evolution of a cell to another cell.

$P-1$ cell. A cell which is not a periodic cell is called transient cell.

The fundamental step in the SCM algorithm is the following. The state of the system at time t is no longer described by the state vector $x(t)$, but by the index $\xi(t) \in 0, \ldots, M$ of the cell containing the state vector

$$\xi(t) = j \iff x(t) \in cell j.$$

As an example of SCM for the treatment of multi-objective optimization problems, we present the following paper:

- Hernandez, Yousef, Yousef, Wei, Schütze and Jian-Qiao Sun introduce the simple cell mapping method for the multi-objective optimal time domain design of feedback controls for linear systems with or without time delay [15]. The authors consider two feedback control design problems to demonstrate their method: a linear quadratic regulator based approach with the weighting matrices as design parameters, and a directed optimization with feedback control gains as design parameters.

2.4 Evolutionary algorithms

An evolutionary algorithm (EA) is a heuristic optimization algorithm using techniques bio-inspired by species evolution such as mutation, recombination and natural selection, in order to find an optimal individual for the given problem [4].

At the beginning of the algorithm, a group of individuals (called population) is created, then in each iteration of the EA (called generation) the algorithm produces a new population via reproduction (or recombination) and mutation, in order to converge to a result the EA uses a natural selection, which selects the 'best' individual of the current population and tries to preserve its genes and makes small modifications by any via (exploitation) or makes mutations on some individuals, in order to get a better individual than the actual 'best' (exploration).

These algorithms are widely used because they produce a 'good' approximation of the Pareto set and Pareto front, without using any extra information and they cut the search space giving us a save of function calls, i.e., total time of execution, which can be a decisive factor.

In the following, we present some EA approaches for the treatment bilevel optimization problems:

- Deb and Sinha address certain intricate issues related to solving multi-objective bilevel programming problems [8]. They also present challenging test problems and propose a viable and hybrid evolutionary-cum-local-search based algorithm as a solution methodology. The population sizing and termination criteria are made self-adaptive, so that no additional parameters need to be supplied by the user. The study indicates a clear niche of evolutionary algorithms in solving such difficult problems of practical importance compared to their usual solution by a computationally expensive nested procedure. The study opens up many issues related to multi-objective bilevel programming.
- Ruuska and Miettinen propose a procedure to construct evolutionary bilevel optimization algorithms based on recent theoretical advances that have

established connections between bilevel optimization and multi-objective optimization [31]. In the proposed procedure, a new algorithm is defined by integrating an evolutionary multi-objective optimization algorithm with a partial order that is compatible with bilevel optimization.

- Sinha, Malo and Deb propose a procedure for designing controlled test problems for bilevel single-objective optimization [39]. The proposed procedure is flexible such that the different complexities present in a bilevel problem can be controlled independently of each other. Also the authors provide a test suite of 12 test problems, which consists of 8 unconstrained and four constrained problems.

- Deb and Sinha propose five test problems, which are scalable in terms of the number of variables and objectives, and which enable researchers to evaluate different phases of a bilevel problem solving task [7]. The test problem construction procedure is interesting and may motivate other researchers to extend the idea to develop further test problems.

- Gupta, Kelly, Ehrgott and Bickerton describe an Evolutionary Algorithm which is presented to solve the problem of the design parameters of the resin transfer molding (RTM) and compression RTM (CRTM) that are popular methods for high volume production of superior quality composite parts [14]. The parameters must be carefully chosen in order to reduce cycle time, capital layout and running cost, while maximizing the final part quality. These objectives are principally governed by the filling and curing phases of the manufacturing cycle, which are strongly coupled in the case of completely non-isothermal processing. Independently optimizing either phase often leads to conditions that adversely affect the progress of the other. In light of this fact, this work models the complete manufacturing cycle as a static Stackelberg game with two virtual decision makers (DMs) monitoring the filling and curing phases, respectively. The model is implemented through a Bilevel multi-objective Genetic Algorithm

(BMOGA), which is integrated with an Artificial Neural Network (ANN) for rapid function evaluations. The obtained results are thus efficient with respect to the objectives of both DMs and provide the manufacturer with a diverse set of solutions to choose from.

2.5 Descent directions

A descent direction is a vector $\nu \in \mathbb{R}^n$, if a descent direction ν is given at a point x, a further candidate solution x_{new} that dominates x can easily be found by a line search, i.e., by setting

$$x_{new} = x + t\nu, \tag{2.16}$$

where $t \in \mathbb{R}_+$ is a suitable step size.

The solution of this kind of problems would give as result a curve of dominated points, i.e., the new point dominates the previous one.

In the following, we present several methods which use this idea to find a descent direction ν.

- Lara proposes the simplest bi-objective descent direction [23], which is one way to combine two gradients to obtain a descent direction by a vector sum

  $$v(x_0) = -(\frac{\nabla f_1(x)}{||\nabla f_1(x)||} + \frac{\nabla f_2(x)}{||\nabla f_2(x)||}), \text{for } \nabla f_i(x) \neq 0 \tag{2.17}$$

  where $||\cdot|| = ||\cdot||_2$,$x \in \mathbb{R}^n$ and v is a descent direction at x_0.

  However, this approach cannot be generalized to more than two objective functions.

- Schütze, Lara and Coello propose the following descent [36]: assume a point $x_0 \in \Omega$ is given as well as a vector $d \in \mathbb{R}^k$ representing a desired direction in objective space.

  $$J(x)v = d,$$

where $v \in \mathbb{R}^n$ is a search direction in parameter space,$|| \cdot || = || \cdot ||_2$ and $J(x)$ is the jacobian matrix. wich is defined by

$$J(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x) & \dots & \frac{\partial f_1}{\partial x_n}(x) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_k}{\partial x_1}(x) & \dots & \frac{\partial f_k}{\partial x_n}(x) \end{bmatrix} \tag{2.18}$$

the authors propose that v can be computed by solving a system of linear equations. Since typically the number of parameters is higher than the number of objectives, the system of equations is underdetermined, which implies that its solution is not unique. To deal with this, the problem can be formulated as:

$$v = J(x_0)^+ d,$$

where $J(x_0)^+$ denotes the pseudo inverse of the Jacobian $J(x_0) \in R^{k \times n}$. Further, we can solve the following initial value problem (IVP):

$$\begin{aligned} x(0) &= x_0 \in \mathbb{R}^n \\ \dot{x}(m) &= \nu_\alpha(x(m)), t > 0. \end{aligned} \tag{2.19}$$

- Schäffler, Schultz and Weinzierl [32] propose the following descent direction:

$$q(x) = \sum_{i=1}^{k} \hat{a} \nabla f_i(x),$$

where $q : \mathbb{R}^n \to \mathbb{R}^n$ and $\hat{a}$ is a solution of

$$\min_{\alpha \in \mathbb{R}^k} \{ || \sum_{i=1}^{k} \alpha_i \nabla f_i(x) ||_2^2, \alpha_i \geq 0, i = 1, \dots, k, \sum_{i=1}^{k} \alpha_i = 1 \},$$

where ∇f_i is the gradient of the ith component of the ith objective function and k is the number of objectives.

From this we have that either $q(x) = 0$ or $-q(x)$ is a descent direction for all the objective functions; hence, each x with $q(x) = 0$ fulfills the first-order necessary conditions for Pareto optimality.

- Fliege and Svaiter [13] propose the following descent direction:

$$f_x(v) = max(Av)_i, i = 1, \dots, m,$$

where $f_x : \mathbb{R}^n \to \mathbb{R}$. We can see that f_x is convex and positive homogeneous. Using this function the authors propose the following problem:

$$\begin{gathered} \min f_x(v) + \frac{1}{2}||v||^2 \\ \text{subject to } v \in \mathbb{R}^n. \end{gathered}$$

From this we have that, if x is Pareto optimal, then $v(x) = 0$. If it is not the case then v is a descent direction.

Chapter 3

Parallel Simple Cell Mapping for Multi-objective optimization problems

In this chapter we focus on the case of SCM for the treatment of MOPs.

As in Equation 2.15 the total cell number in the system grows exponentially as the number of dimensions increases. This represents a problem due the limited amount of memory and processor time.

In order to overcome this problem we can use two different approaches:

- To use parallel computing, in order to process more cells in less time.
- To reduce the search space, in order to process less cells and therefore reduce the computational burden.

In order to choose which path is the best to solve this problem, we can state some SCM properties. Hernández Castellanos propose an algorithm [17], which its most expensive part is the computation of the dynamical system. In this part, at least two function evaluations are performed inside the cell, one of those is for the evaluation of the center point and at least one evaluation for the computation of the step size.

In the cell evolution part, the mapping C (SCM) is given by a point-wise iteration $x_j = x_i + t\nu$:

Where:

x_i: Center point of the current cell

x_j: Final point

t: Step size

ν: Descent direction

Since x_i, t and ν can be known locally, in other words, the evolution of a cell can be done without any other information of another cell, therefore the mapping function C, is applied on each cell in the system , that means a Single Instruction Multiple Data computer could do this task in a parallel way and also means that we can use data parallelism.

If we have a great amount of processors ($p = \infty$), the theoretical acceleration given by Amdhal's law (see Equation 2.9) and the number of cells in the system (see Equation 2.15) is $p_s = \frac{1}{M}$, thus the computational time is given by only one evaluation.

In conclusion, we can apply parallel techniques to increase the number of cells that can be processed in a certain amount of time without changing the algorithm, therefore we still get all the information that SCM provide, but in other hand, the computational complexity remains the same.

If we choose to cut the search space, we use algorithms that has been tested, like refinement or subdivision techniques, in this way we can make at the beginning bigger cells, then when the algorithm reach a point of quality, take the best results and refine the cell until it reach a desired quality, in that way we can reduce the search space of the problem and thus the computational time.

To cut the search space can be very useful for higher dimensions, but we lost a lot of information of the dynamical system that can be useful.

These two approaches are different, but we can apply both in order to get a better algorithm. This task has to be performed carefully, because some techniques needs to be performed sequentially making the overall algorithm less efficient in the context of parallel computing, thus in practice this algorithm will be inefficient due to the overhead of threads control.

3.1 Parallel Simple Cell Mapping

The algorithm [16] can be divided into three principal parts:

- Creation of the cells: where the information (center point, boundaries, index, etc.) of each cell is computed.
- Evolution of the system: where the SCM is processed.
- Post-processing: where it is performed extra algorithms in order to obtain the set of approximate solution.

Algorithm 1 works on the GPU side, this algorithm creates the set of cells together with their center points to be processed by the Algorithm 2. The key factor of this algorithm is the function $Getid()$ which must return a unique and valid value, which is the identity id of a cell c_{id} for instance $Getid() = threadIdx.x + blockIdx.x * blockDim.x$, where $threadIdx.x$, $blockIdx.x$ and $blockDim.x$ are built-in GPU's variables. Each cell has a center point cp and a function value fv .

Algorithm 1 Creation of the cell space

```
Require: Function $F$
1: $id \leftarrow Getid()$
2: $c_{id}.cp \leftarrow$ Center point of the cell.
3: $c_{id}.fv \leftarrow f(c_{id}.cp)$
```

Once the algorithm created on GPU memory all the required information we can proceed with the evaluation of each cell (Algorithm 2). As for Algorithm *Creation of the cell space*, we need to calculate a unique and valid id for the cell. Then, we calculate the mapping defined by the underlying dynamical system for the cell c_{id}.

Based on this, we decide if the cell is mapped into another cell $\tilde{c}$ (i.e., a new vector is found dominating the center point of the current box which is located in cell $\tilde{c}$) or the cell is mapped to itself.
It is important to note that the division of the problem is over the data set (each cell is processed individually on each core), thus the theoretical acceleration is given by the numbers of cells m_c in the system

$$S(n) = \frac{1}{\frac{1}{m_c} + \frac{1}{n}(1 - \frac{1}{m_c})}. \tag{3.1}$$

Algorithm 2 Evolution of the system

```
Require: Number $tot$ of total cells in the system and the sizes $h$ per dimension
1: $id \leftarrow Getid()$
2: Calculate the decent direction $v$
3: $t \leftarrow norm(h)$
4: $aux \leftarrow f(c_{id}.cp + t * v)$
5: while $c_{id}.fv \prec aux$ do
6:     $t \leftarrow t/2$
7:     $aux \leftarrow f(c_{id}.cp + t * v)$
8: end while
9: $c_{id}.nextCell \leftarrow i$ s.t. $aux \in c_i$
```

In this way, we can proceed with the analysis (see Algorithm 3) in order to mark the candidate cells where the non-dominated set might be located in. Algorithm *Analysis of the dynamical system* goes through all cells and checks if they belong to the p-group. In principle, only 1-groups are of interest (i.e., self-mapping cells). Note, however, that by discretization of the cell mapping ansatz also p-groups with $p > 1$ can be generated that are of interest (the most common ones are 2-groups that perform a 'flipping' around a (local) solution). The algorithm marks all p-groups as candidates when their period is less or equal to a maximum value mp i.e. $p \leq mp$.

In order to select the Pareto set, a parallel dominance test has to be performed using the center points of all candidate cells, see Algorithm 4. Cells those center points are not dominated by any other center points of the candidate set are stored as part of the (obtained) Pareto set. The Pareto front can then e.g. be approximated

Algorithm 3 Analysis of the dynamical system

```
Require: Total cells $tot$ in the system and the maximal period $mp$.
1: $id \leftarrow Getid()$
2: $i \leftarrow 0$
3: $c = i \leftarrow c_{id}.nextCell$
4: while $c_{id} \neq c_i$ and $i < mp$ do
5:     $c = i \leftarrow c_{id}.nextCell$
6: end while
7: if $c_{id} = c_i$ then
8:     mark $c_{id}$.
9: end if
```

via considering the function values of the center points.

Algorithm 4 Dominance Test

```
Require: Total candidate cells $tot$ and a set $C$ of candidates cells.
1: $id \leftarrow Getid()$
2: for all $c \in C$ do
3:     if $c_{id} \prec c$ then
4:         unmark the cell.
5:     end if
6: end for
7: if $c_{id}$ is marked then
8:     Save $c_{id}$
9: end if
```

Finally, Algorithm 5 states the pseudo code of the parallel version of the SCM.

3.1.1 Numerical Results

In the following we investigate the acceleration obtained via parallel SCM compared to its sequential counterpart. Table 3.1 shows the execution times for the creation and the evaluation algorithm as well as for the complete procedure. In all cases we have used $n = 10$ for the dimension of the parameter space. For the computations, we have used an Alienware Mx17 R4 laptop with 8GB RAM and the following characteristics: (i) a Nvidia gtx 680m GPU with 1344 CUDA cores and a 720 MHz processor clock, (ii) an Intel Core i7-3720QM CPU with a clock speed of 2.6 GHz and a maximal turbo frequency of 3.6 GHz.

```
Algorithm 5 parallel SCM
Require: The objective functions vector $F$, total cells $tot$ in the system, the sizes $h$
    per dimension and their limits $lim$
Ensure: Approximation of the Pareto set and Pareto front of the MOP.
1: Creation of the cell space($F$,$tot$,$h$,$lim$)
2: Evolution of the system($f$,$tot$,$h$,$lim$)
3: Analysis of the dynamical system($tot$,2)
4: DominanceTest($tot$)
```

Table 3.1: Computational times and speedups obtained by SCM and pSCM on the benchmark models. All the times are in milliseconds.

	Creation		Evaluation		Total		Total acceleration
	parallel	sequential	parallel	sequential	parallel	sequential	
$ZDT1$	150.78	1367.52	642.99	3861.02	1345.11	5461.8	4.06x
$ZDT2$	157.94	2025.54	646.44	4318.52	1270.69	6989.44	5.50x
$ZDT3$	144.18	1705.36	642.99	3861.02	1326.13	6244.94	4.70x
$ZDT4$	274.72	3235.67	677.76	6727.01	1506.64	10216.8	6.78x
$ZDT6$	216.87	2740.5	851.26	7047.48	1688	10101.29	5.98x

The total acceleration for these seven examples ranges from a speedup of four to nearly seven. This is on the one hand not that overwhelming. Due to the overhead associated with transitions between GPU and CPU, the practical acceleration of each problem is (much) below the theoretical one. On the other hand, we note that parallel computing allows for computational times of 1.3 to 1.7 seconds which is more than reasonable for the global solution of problems of that dimension. Further, it has to be noted that the evaluation time for each of the chosen academic models is quite low (less than a millisecond per function evaluation). For real-world applications, for which the cell mapping techniques are originally designed, such evaluation times are not realistic, but can range in minutes or even hours as e.g. for airfoil design ([20, 26]). In order to artificially increase the cost of our benchmark functions without defining new ones, we evaluate each function m times in each cell. The next tables show the total acceleration obtained for the values $m = 10$ (Table 3.2), 100 (Table 3.3) and 1000 (Table 3.4) .

The main reason for the difference in the accelerations shown in the tables is due to logarithmic and trigonometric operations. Inside the GPU, these operations are

Table 3.2: Computational times and speedups for $m = 10$ function evaluations to increase the cost of the evaluation. All the times are in milliseconds.

	Creation		Evaluation		Total		Total acceleration
	parallel	sequential	parallel	sequential	parallel	sequential	
$ZDT1$	216.78	4100.29	885.33	9544.66	1653.45	13878.21	8.39x
$ZDT2$	908.58	8305.58	947.844	14753.33	2353.334	23268.37	9.88x
$ZDT3$	855.21	5561.39	870.34	12059.91	2309.36	17837.43	7.72x
$ZDT4$	1785.53	23592.76	1508.6	36939.37	3841.77	60789.04	15.82x
$ZDT6$	1271.08	19319.85	1593.42	40400.23	3535.89	60045.2	16.98x

Table 3.3: Computational times and speedups for $m = 100$ function evaluations to increase the cost of the evaluation. All the times are in milliseconds.

	Creation		Evaluation		Total		Total acceleration
	parallel	sequential	parallel	sequential	parallel	sequential	
$ZDT1$	773.86	28397.96	2377.93	57299.13	3690.73	85962.74	23.29x
$ZDT2$	7336.26	64829.31	4033.77	104667.53	11888.89	169704.77	14.27x
$ZDT3$	6873.71	39873.82	2882.07	77183	10274.94	117319.35	11.41x
$ZDT4$	15543.47	206302.48	9310.72	306838.34	25409.73	513415.04	20.20x
$ZDT6$	10311.19	161705.26	8578.83	335403.28	19511.11	497455.11	25.49x

Table 3.4: Computational times and speedups for $m = 1000$ function evaluations to increase the cost of the evaluation. All the times are in milliseconds.

	Creation		Evaluation		Total		Total acceleration
	parallel	sequential	parallel	sequential	parallel	sequential	
$ZDT1$	6031.4	270900.7	17609.6	532799.7	24198.26	803979.4	33.22x
$ZDT2$	71108.5	631137.9	34965.96	1001487.8	106581.5	1632867.0	15.32x
$ZDT3$	67032.0	383867.53	23360.0	729958.5	90903.2	1114083.9	12.25x
$ZDT4$	152470.2	2039935.3	87354.85	3056862.0	240392.0	5097091.6	21.20x
$ZDT6$	100382.3	1588335.8	78485.2	3282948.5	179502.0	4871649.8	27.13x

performed in hardware units called *Special Functions Units* which are fewer than the number of cores in the GPU. However, as can be seen speedups of more than 30 can be obtained. For instance, the computational time for ZDT6 is about 3 minutes when using pSCM, while the sequential SCM needs 81 minutes for the same computation.

3.2 Parallel Subdivision Simple Cell Mapping

The above algorithms constitutes the required steps to perform the parallel cell mapping for a fixed discretization of the parameter space. Note, however, that even if we achieve a great performance due to the GPUs, we still face the same dimensionality problem as for the sequential algorithm, namely that the number of cells (and thus the overall cost of the algorithm) increases exponentially with the number of dimensions. which will eventually overflow the memory capabilities of any computer. Here, we follow the suggestion made in [24] and perform the cell mapping in an iterative way on the set of candidate cells. In this way, cell mapping gets combined with subdivision techniques presented in [10, 19, 37].

The new algorithm must store the boundaries of each cell, therefore Algorithm 1 must be redefined. Algorithm 6 stores the boundaries of each cell.

Algorithm 6 Creation of the cell space with boundaries

```
Require: Function $F$, total cells $tot$ in the system, sizes $h$ per dimension, and limits $lim$ per dimension.
1: $id \leftarrow Getid()$
2: $c_{id}.cp \leftarrow$ Center point of the cell.
3: $c_{id}.lim \leftarrow$ local boundaries,
4: $c_{id}.fv \leftarrow f(c_{id}.cp)$
```

Algorithm 7 presents the overall iterative parallel version of SCM for the treatment of multi-objective optimization problems using the Algorithm 6. The main idea is to make a gross SCM trying to use all the available GPU processors. Then, the best points selected by the dominance test constitute the new search space. This process, cell mapping and selection, is iterated until the desired depth md of the search is reached.

Algorithm 7 pSCM

```
Require: Total cells $tot$ in the system, the sizes $h$ per dimension and their limits
    $lim$, the depth $d$ of the search, and the maximal depth $md$.
Ensure: Approximation of the Pareto set and Pareto front of the MOP.
 1: $cs \leftarrow \emptyset$
 2: if $d = md$ then
 3:     Store candidate set $cs$.
 4: else
 5:     Create of the state space with boundaries($f$,$tot$,$h$,$lim$)
 6:     Evolution of the system($f$,$tot$,$h$,$lim$)
 7:     Analysis of the dynamical system($tot$,2)
 8:     $cs \leftarrow$ DominanceTest($tot$)
 9:     for all $c_i \in cs$ do
10:         $h \leftarrow h/tot$
11:         pSCM($f$,$tot$,$h$,$c_i.lim$,$d+1$,$md$)
12:     end for
13: end if
```

3.2.1 Numerical Results

Here we present some numerical results of the novel parallel cell mapping algorithm and discuss the speedup against its sequential counterpart. For the problems, we have chosen to take the bi-objective ones from the ZDT [43] and DTLZ [9] benchmark suites. These problems have different characteristics (e.g., convex, concave or disconnected Pareto front, uni-modal or multi-modal functions, Pareto front within or at the boundary of the domain) and are widely used to demonstrate the capability of an algorithm to compute the entire Pareto front of a given MOP. In all cases we have used $n = 10$ for the dimension of the parameter space. For the computations, we have used an Alienware Mx17 R4 laptop with 8GB RAM and the following characteristics: (i) a Nvidia gtx 680m GPU with 1344 CUDA cores and a 720 MHz processor clock, (ii) an Intel Core i7-3720QM CPU with a clock speed of 2.6 GHz and a maximal turbo frequency of 3.6 GHz. The design parameters of pSCM for each problem can

Table 3.5: Number of restarts and divisions per coordinate direction for the MOPs considered in this study.

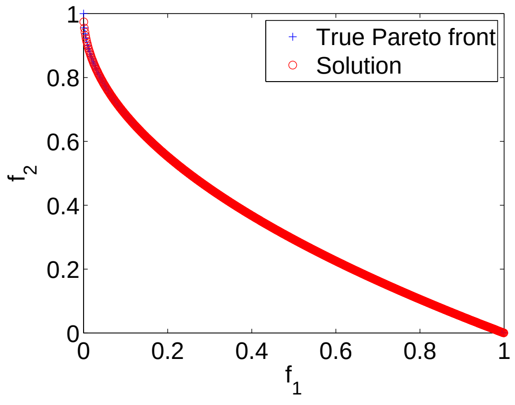
	restarts	divisions per coordinate
$ZDT1$	12	[2 2 2 2 2 2 2 2 2 2 2]
$ZDT2$	12	[2 2 2 2 2 2 2 2 2 2 2]
$ZDT3$	6	[5 3 3 3 3 3 3 3 3 3 3]
$ZDT4$	4	[5 5 5 5 5 5 5 5 5 5]
$ZDT6$	12	[2 2 2 2 2 2 2 2 2 2 2]
$DTLZ1$	12	[2 2 2 2 2 2 2 2 2 2 2]
$DTLZ2$	6	[5 3 3 3 3 3 3 3 3 3 3]

Table 3.6: Obtained Δ_2 values of the approximations obtained by pSCM and the true Pareto fronts.

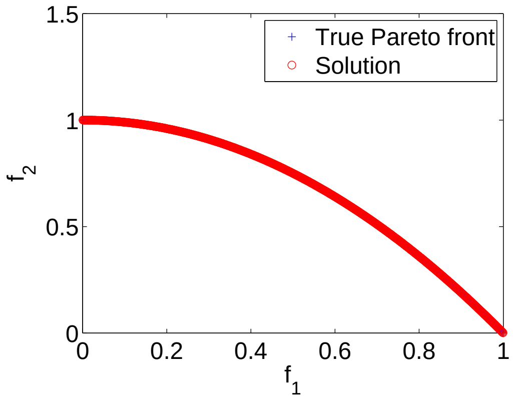
	Δ_p value
$ZDT1$	0.0043
$ZDT2$	0.0043
$ZDT3$	0.0035
$ZDT4$	0.0043
$ZDT6$	0.0037
$DTLZ1$	0.0011
$DTLZ2$	0.0046

be found in Table 3.5.

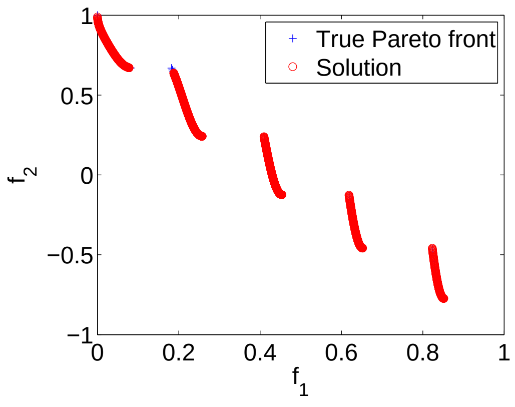
Figure shows the numerical approximations of the Pareto fronts of the ZDT functions plus the true Pareto fronts and Figure the respective ones for the DTLZ functions. Apparently, in all cases the approximations are nearly identical to the true Pareto sets and can certainly be considered to be 'good enough' from the practical point of view. This observation is underlined by Table 3.6 which shows the obtained Δ_p values (for $p = 2$) of the obtained approximations and the true Pareto fronts. The indicator Δ_p [34] can be viewed as an averaged Hausdorff distance between two sets. Since all values are close to zero, this indicates a good approximation quality of the outcome sets.



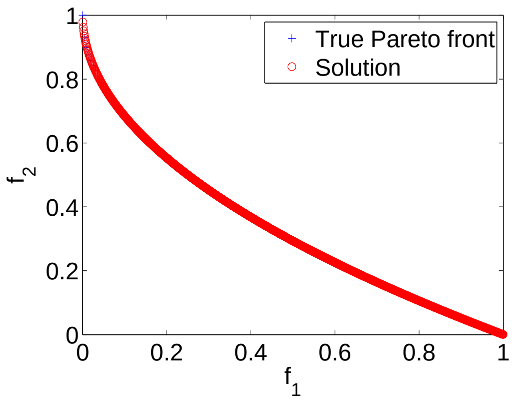
(a) ZDT1



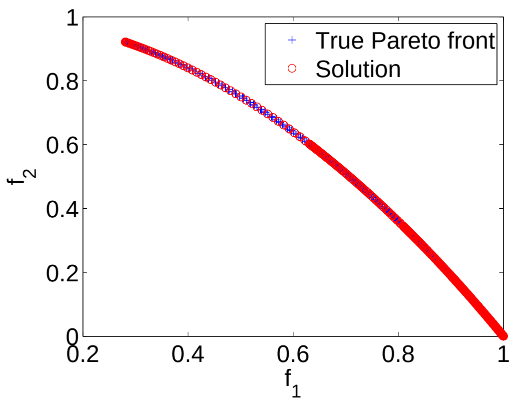
(b) ZDT2



(c) ZDT3

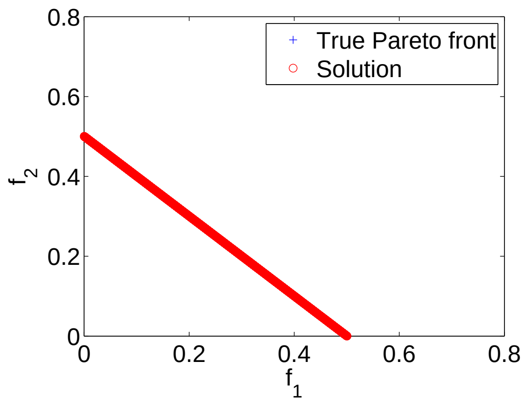


(d) ZDT4

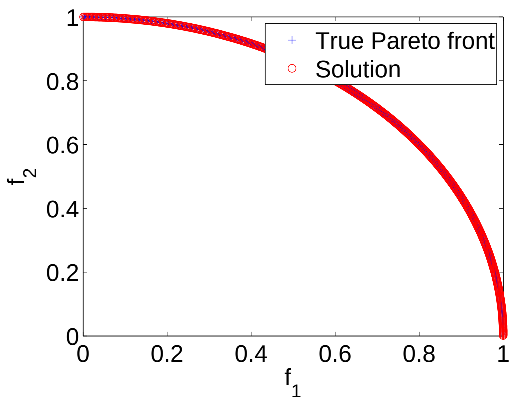


(e) ZDT6

Figure 3.1: True Pareto front and approximations for ZDT benchmark suite.



(a) DTLZ1



(b) DTLZ2

Figure 3.2: True Pareto front and approximation for DTLZ benchmark suite with 2 objectives.

3.3 Cell Control

We have to consider that as the number of objectives increases, the number of candidate solution also increases. This may lead to an overflow of memory and workload as the number of iteration increases. For this reason we propose to use the external archive ArchiveUpdate2 [35] with $\epsilon = 0$. This archive allows us to define a value δ which defines a desired distance between solutions in objective space. In this case, we define an appropriate value for δ. That is performed as follows:

$$\delta = \frac{\max_{i \in k} F_i(cell) - \min_{i \in k} F_i(cell)}{max\_arch},$$

Algorithm 8 presents the overall iterative pSCM for the treatment of multi-objective optimization problems. The main idea is to make a gross SCM trying to use all the available GPU processors. Then, cells selected after the archive constitute the new search space. This process, cell mapping and selection, is iterated until the desired depth md of the search is reached.

Algorithm 8 pSCM 2nd version

```
Require: Total cells $tot$ in the system, the sizes $h$ per dimension and their limits
    $lim$, the depth $d$ of the search, and the maximal depth $md$.
Ensure: Approximation of the Pareto set and Pareto front of the MOP.
 1: $cs \leftarrow \emptyset$
 2: if $d = md$ then
 3:     Store candidate set $cs$.
 4: else
 5:     Creation of the cell space with boundaries($f$,$tot$,$h$,$lim$)
 6:     Evolution of the system($f$,$tot$,$h$,$lim$)
 7:     Analysis of the dynamical system($tot$,2)
 8:     $cs \leftarrow$ ArchiveUpdateTight2($tot$, $\delta$)
 9:     for all $c_i \in cs$ do
10:         $h \leftarrow h/tot$
11:         pSCM($f$,$tot$,$h$,$c_i.lim$,$d+1$,$md$)
12:     end for
13: end if
```

3.3.1 Numerical Results

Here we present some numerical results of the 2nd version of pSCM and discuss the speedup against its sequential counterpart. To measure the approximation quality we have used the averaged Hausdorff distance Δ_2 which is common for the comparison of different algorithms.

For the problems, we have chosen to take DTLZ [9] benchmark suites for $k = 2, \ldots, 3$. These problems have different characteristics (e.g. concave or disconnected Pareto front, uni- or multi-modal functions, Pareto front within or at the boundary of the domain) and are widely used to demonstrate the capability of an algorithm to compute the entire Pareto front of a given MOP.

In all cases, we have used $n = 10$ for the dimension of the parameter space. For the computations, we have used an Alienware Mx17 R4 laptop with 8GB RAM and the following characteristics: (i) a Nvidia gtx 680M GPU with 1344 CUDA cores and a 720 MHz processor clock, (ii) an Intel Core i7-3720QM CPU with a clock speed of 2.6 GHz and a maximal turbo frequency of 3.6 GHz.

For the problems with $k = 2$, Table 3.7 contains the initial parameters of the

Table 3.7: Number of restarts and divisions per coordinate direction for the MOPs considered in this study.

	restarts	divisions per coordinate
$DTLZ1$	5	[3 3 3 3 3 3 3 3 3 3 3]
$DTLZ2$	5	[3 3 3 3 3 3 3 3 3 3 3]
$DTLZ3$	5	[3 3 3 3 3 3 3 3 3 3 3]
$DTLZ4$	5	[3 3 3 3 3 3 3 3 3 3 3]
$DTLZ6$	5	[3 3 3 3 3 3 3 3 3 3 3]
$DTLZ7$	5	[3 3 3 3 3 3 3 3 3 3 3]

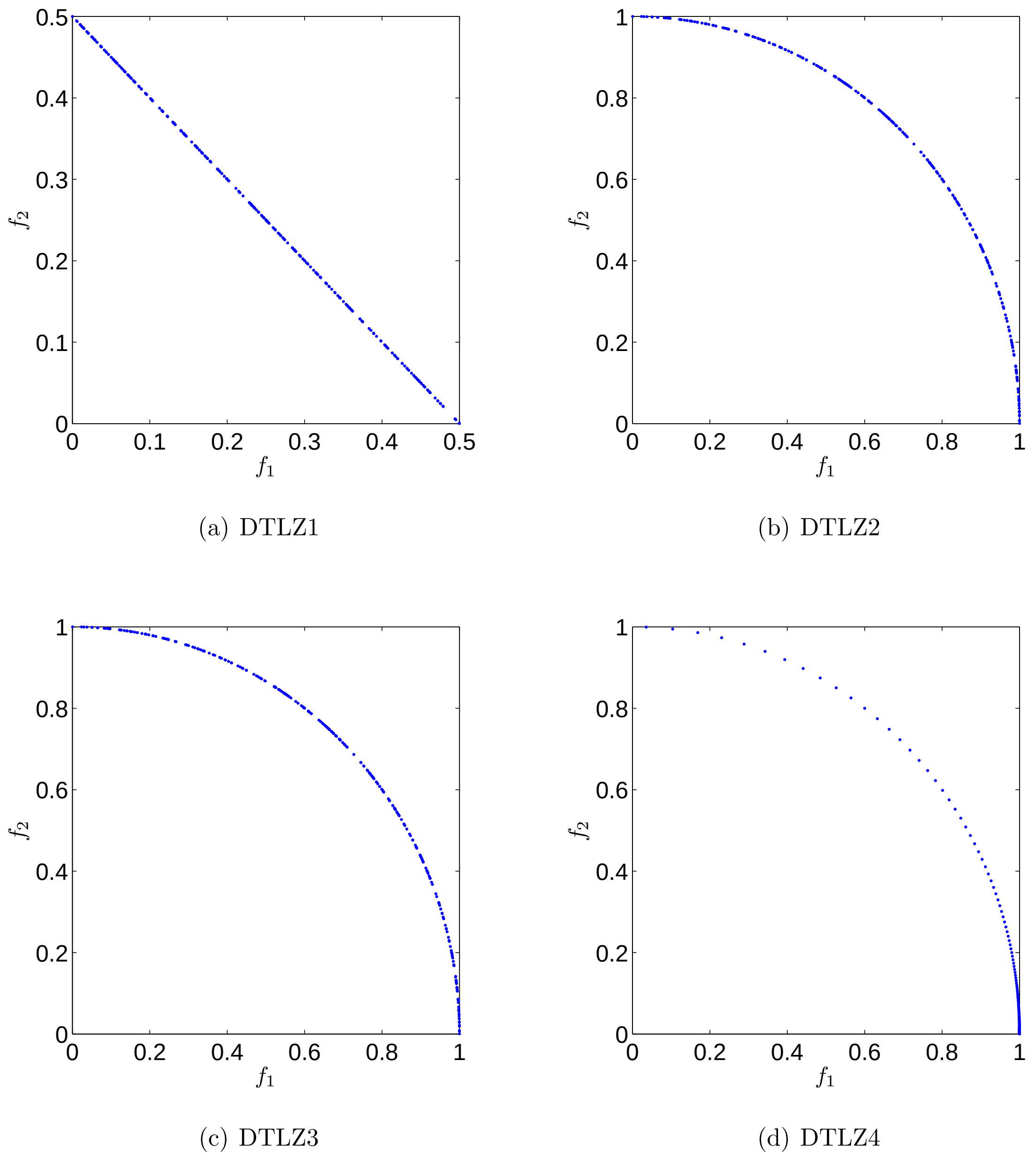
algorithm. Next, we set the limit to 300 solutions for the archive.

For the problems with $k = 3$, we used a initial grid of the same size as for $k = 2$. We used seven restarts. We set the limit to 500 solutions for the archive.

Figures 3.3, 3.4,3.5 and 3.6 show the results for DTLZ functions for $k = 2$ and $k = 3$ respectively. As we can see from the pictures, in all cases we obtain a fine grain representation of the Pareto front. This can be confirmed in Table 3.9, which shows the Δ_2 values computed by pSCM. The reference fronts where taken from [4] This values show that we are able to obtain a good approximation to the real Pareto front as the values are close to zero.

Table 3.8 shows the speedups of the pSCM compared with SCM for $k = 3$. The table shows that we can obtain a speedup close to $10x$. As we can see, this speedup is far from theoretical values described above. This is due to the overhead associated with transitions between GPU and CPU. Further, it has to be noted that the evaluation time for each of the chosen academic models is quite low (less than a millisecond per function evaluation).

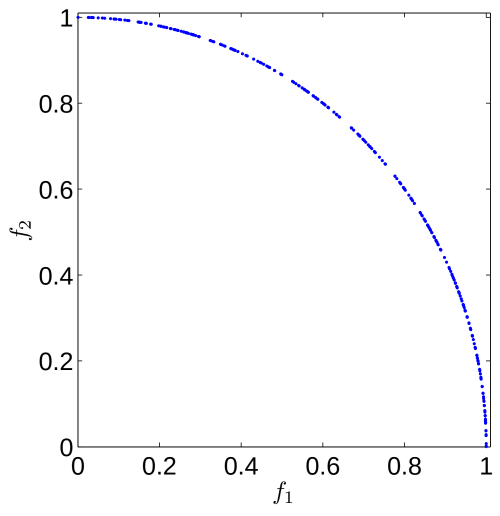
In order to increase the cost of our benchmark functions without defining new ones, we evaluate each function m times in each cell. Since each cell is treated individually (we upload one cell to the GPU, then we subdivide it and finally we process it), we can consider the time for one cell and then multiply it by the number of cells processed. This is valid for both sequential and parallel SCM. Thus, we use this method to measure the speedups for benchmark functions with additional time



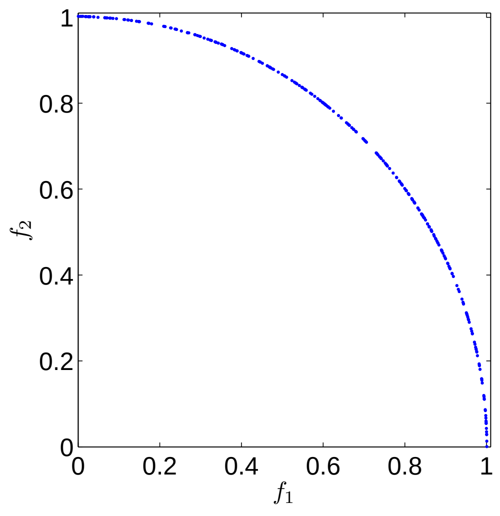
(a) DTLZ1 (b) DTLZ2

(c) DTLZ3 (d) DTLZ4

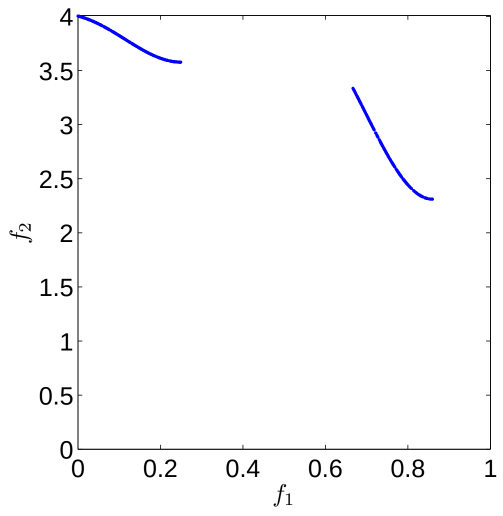
Figure 3.3: Pareto fronts of DTLZ functions for $k = 2$ obtained by pSCM.



(a) DTLZ5

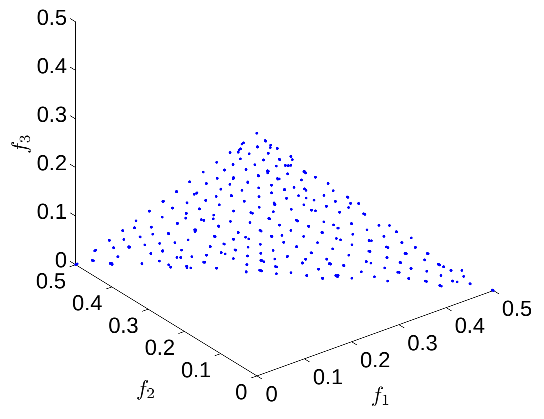


(b) DTLZ6

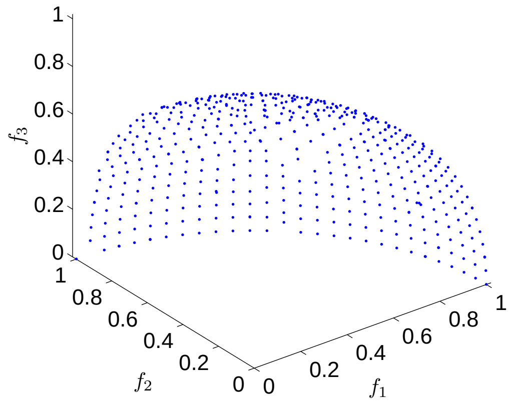


(c) DTLZ7

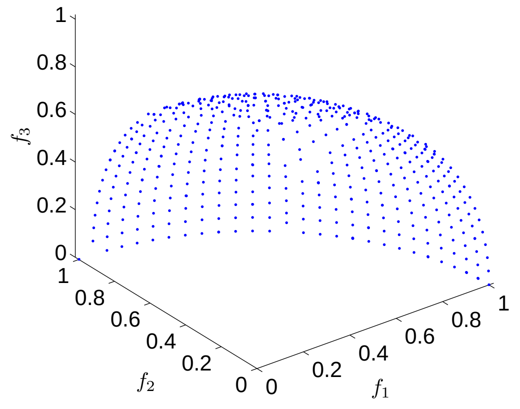
Figure 3.4: Pareto fronts of DTLZ functions for $k = 2$ obtained by pSCM.



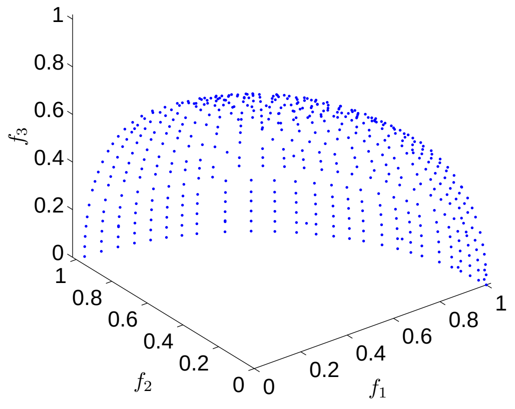
(a) DTLZ1



(b) DTLZ2

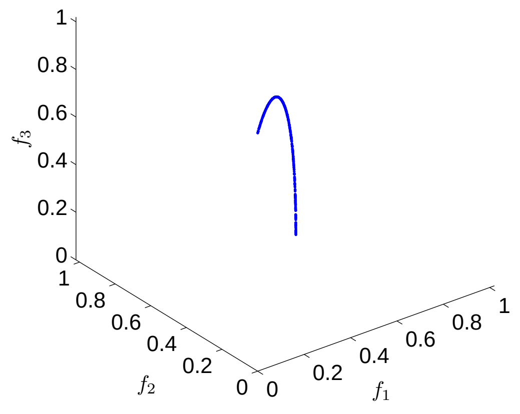


(c) DTLZ3

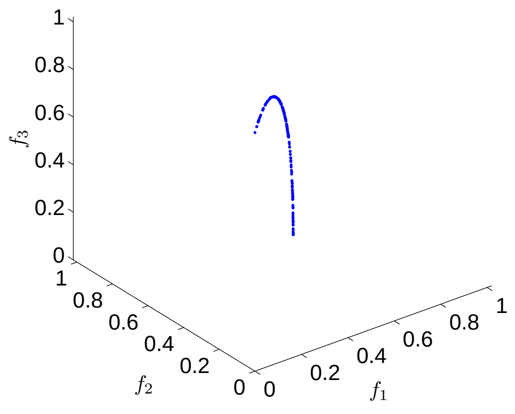


(d) DTLZ4

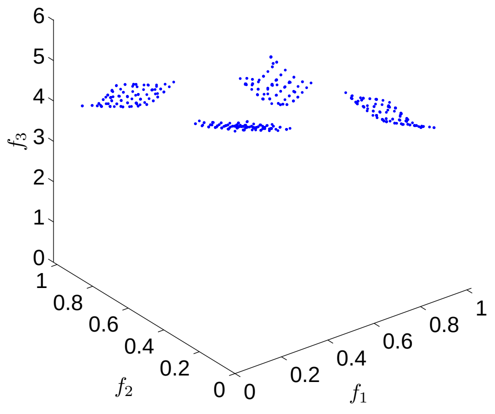
Figure 3.5: Pareto fronts of DTLZ functions for $k = 3$ obtained by pSCM.



(a) DTLZ5



(b) DTLZ6



(c) DTLZ7

Figure 3.6: Pareto fronts of DTLZ functions for $k = 3$ obtained by pSCM.

Table 3.8: Comparison times and accelerations of DTLZ functions for $k = 3$ for sequential and parallel SCM in milliseconds.

Problem	Parallel	Serial	Acceleration
DTLZ1	257923	3125320	12.1172598023
DTLZ2	255774	3042160	11.8939376168
DTLZ3	258532	3095730	11.9742623737
DTLZ4	178957	2125210	11.8755343462
DTLZ5	100376	1161780	11.5742807046
DTLZ6	251401	2963050	11.7861504131
DTLZ7	217742	2709310	12.4427533503

Table 3.9: Obtained Δ_2 values of the approximations obtained by pSCM and the true Pareto fronts.

Problem	$k = 2$	$k = 3$
DTLZ1	0.0021	0.0131
DTLZ2	0.0039	0.0254
DTLZ3	0.0039	0.0256
DTLZ4	0.0122	0.0384
DTLZ5	0.0043	0.0024
DTLZ6	0.0041	0.0061
DTLZ7	0.0388	0.0466

cost.

Tables 3.10 and 3.11 shows the total acceleration obtained for $k = 2$ and $k = 3$ respectively. The values shown correspond to $m = 10, 100$ and 1000 (even for $m = 1000$ the evaluation time for each function is still much less than one second). We can see from the acceleration values that as the function cost increases, we can achieve grater accelerations. This can be seen in Figures 3.7 and 3.8, which show the acceleration graphics for $k = 2$ and $k = 3$ respectively.

Table 3.10: Acceleration of DTLZ functions for $k = 2$ obtained by pSCM for different costs of the function evaluation.

Problem	10x			100x			1000x		
	Parallel	Serial	Acc	Parallel	Serial	Acc	Parallel	Serial	Acc
DTLZ1	127	1940	15.2755	232	7380	31.8103	1320	63230	47.9015
DTLZ2	122	1910	15.6557	219	6320	28.8584	1180	43680	37.0169
DTLZ3	126	1980	15.7142	251	7650	30.4780	1513	64410	42.5710
DTLZ4	120	1830	15.25	224	7040	31.4285	1255	55790	44.4541
DTLZ5	122	1750	14.3442	217	6180	28.4792	1198	46470	38.7896
DTLZ6	124	2670	21.5322	241	14470	60.0414	1436	138130	96.1908
DTLZ7	131	1410	10.7633	154	1720	11.1688	504	5670	11.25

Table 3.11: Acceleration of DTLZ functions for $k = 3$ with pSCM for different function costs.

Problem	10x			100x			1000x		
	Parallel	Serial	Acc	Parallel	Serial	Acc	Parallel	Serial	Acc
DTLZ1	143	2120	14.8251	271	7500	27.6752	1524	62050	40.7152
DTLZ2	144	2070	14.375	283	6890	24.3462	1709	56590	33.1129
DTLZ3	148	2290	15.4729	318	8460	26.6037	1987	71360	35.9134
DTLZ4	137	2340	17.0802	292	10670	36.5410	1813	94820	52.3000
DTLZ5	140	1990	14.2142	284	6850	24.1197	1690	54990	32.5384
DTLZ6	145	2750	18.9655	308	14560	47.2727	1901	133400	70.1735
DTLZ7	138	1690	12.2463	212	2590	12.2169	940	9610	10.2234

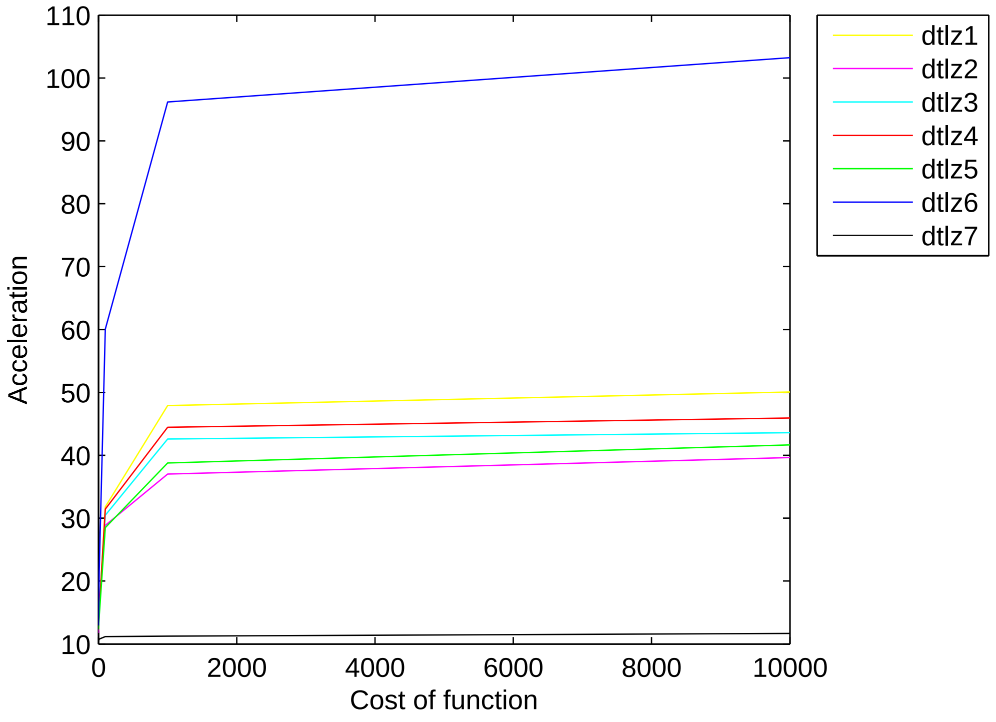


Figure 3.7: Acceleration for DTLZ functions for $k = 2$.

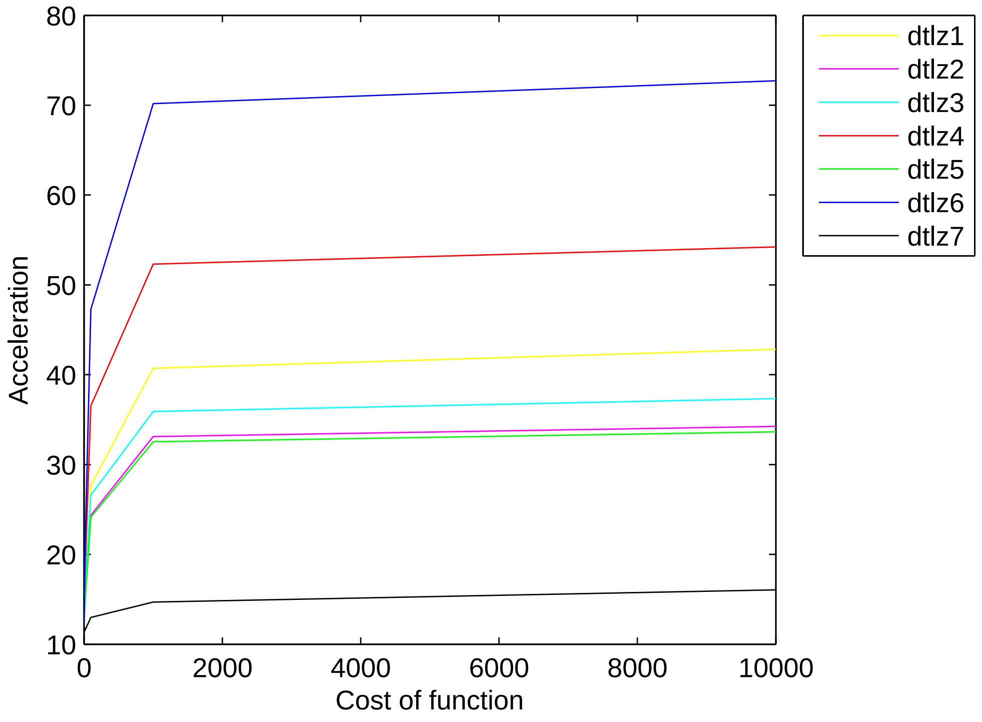


Figure 3.8: Acceleration for DTLZ functions for $k = 3$.

3.4 Approximate Solutions

After the run of the Algorithm 8, we have computed an approximation of the global optimal points. If we remove the archive control technique and the dominance check, we also compute the local optimal points. In this setting, we look for the set of approximate solutions. Algorithm 8 shows the key elements for this algorithm. This algorithm works as follows: Starting in the first pSCM iteration, for each candidate cell we take n samples inside the cell and count how many of those points are non-ϵ-dominated (see Algorithm 10). If the count is equal to the number of samples, we add that box to an archive PQ_ϵ, otherwise, if the count is bigger than zero, we subdivide the cell and do the same procedure and add these new cells to the candidate set. The algorithm finishes when it reaches the number of iterations.

Algorithm 9 ϵ-Dominance

```
Require: P: Pareto front of the problem, ϵ: value of epsilon, y: function evaluation
  .
Ensure: True if y is non-ϵ-dominated.
  for all $p_i \in P$ do
      if $p_i \prec_\epsilon ys$ then
  return False
      end if
      $PQ_\epsilon \leftarrow \emptyset$
  end for
  return True
```

3.4.1 Numerical Results

Next, we show the capabilities of the pSCM to compute not only the global front but also all the local fronts. We show this on the following problem based on the one

Algorithm 10 doNSamplings

```
Require: $f$: Objective functions, $n$: number of samplings, $lim$: cell boundaries, $P$: Pareto front .
Ensure: The number of non-$\epsilon$-dominated elements.
  $i \leftarrow 0$
  $c \leftarrow 0$
  repeat
      $x \leftarrow$ random point $\in lim$
      $y \leftarrow f(x)$
      $c \leftarrow c + \epsilon$-dominance$(P, \epsilon, y)$
      $i \leftarrow i + 1$
  until $i \neq n$
  return c
```

proposed in [30]:

$$\begin{aligned}
&F(x) = (f_1(x), f_2(x)), \text{ where:} \\
&\qquad f_1(x) = (x_1 - t_1(c+2a) + a)^2 + (x_2 - t_2 b)^2 + \delta_t \\
&\qquad f_2(x) = (x_1 - t_1(c+2a) - a)^2 + (x_2 - t_2 b)^2 + \delta_t \\
&\qquad \text{where} \\
&\qquad t_1 = \operatorname{sgn}(x_1) \min\left(\left\lceil \frac{|x_1| - a - c/2}{2a+c} \right\rceil, 1\right), \\
&\qquad t_2 = \operatorname{sgn}(x_2) \min\left(\left\lceil \frac{|x_2| - b/2}{b} \right\rceil, 1\right), \\
&\qquad \delta_t = \begin{cases} 0 & \text{for} \quad t_1 = 0 \text{ and } t_2 = 0 \\ 0.01 & \text{for} \quad t_1 = -1 \text{ and } t_2 = 0 \end{cases}
\end{aligned} \tag{3.2}$$

We used $n = 3$, a grid of $3 \times 3 \times 3$ and 4 subdivision steps. Figure 3.9 shows the result using pSCM, as we can see the method is able to correctly identify all connected components of the Pareto set.

Further, we present a result of approximate solutions on the following problem

Algorithm 11 getEpsilonDominance

```
Require: $C$: main cell of the problem,dd $n$: number of samples, $f$: function, $i$: iterations, $lim$: box constraints, $P$: Pareto front found.
Ensure: $PQ_\epsilon$, $F(PQ_\epsilon)$
  $PQ_\epsilon \leftarrow \emptyset$
  $F(PQ_\epsilon) \leftarrow \emptyset$
  $aux \leftarrow doNSamplings(n, lim, P)$
  if $aux = n$ then
      $PQ_\epsilon \leftarrow PQ_\epsilon \cup c.boundaries$
      $F(PQ_\epsilon) \leftarrow F(PQ_\epsilon) \cup F(c.boundaries)$
      return
  end if
  $C.boundaries \leftarrow$ load the cells of the first iteration
  $C.iteration \leftarrow 0$
  for all $c \in C$ do
      if $c.iteration = i$ then
          $PQ_\epsilon \leftarrow PQ_\epsilon \cup c.boundaries$
          $F(PQ_\epsilon) \leftarrow F(PQ_\epsilon) \cup F(c.boundaries)$
      else
          $aux \leftarrow doNSamplings(n, c.boundaries, P)$
          if $aux = n$ then
              $PQ_\epsilon \leftarrow PQ_\epsilon \cup c.boundaries$
              $F(PQ_\epsilon) \leftarrow F(PQ_\epsilon) \cup F(c.boundaries)$
          else if $aux > 0$ then
              $C^{sub}.boundaries \leftarrow$ subdivide cell $c$ and compute boundaries.
              $C^{sub}.iteration \leftarrow c.iteration + 1$
              $C \leftarrow C \cup C^{sub}$
          end if
      end if
  end for
```

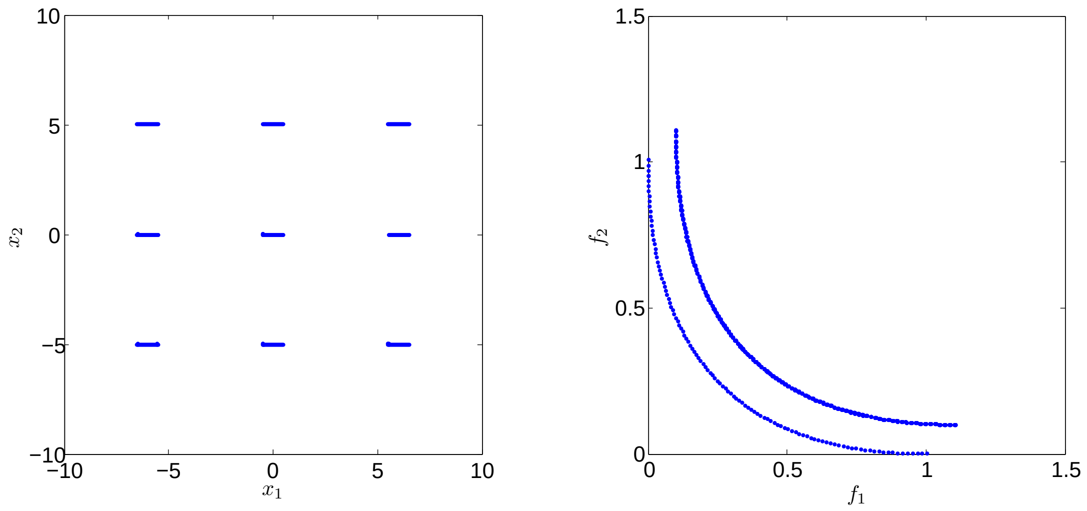
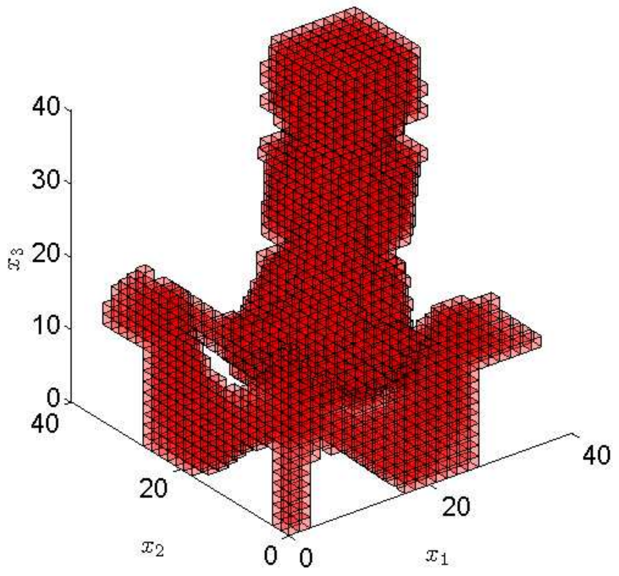


Figure 3.9: Sett of locally optimal solutions of problem 3.2

proposed in [33]:

$$\begin{aligned} &F(x) = (f_1(x), f_2(x)), \text{ where:} \\ &\quad f_1(x) = \sum_{j=1}^{n} x_j, \\ &\quad f_2(x) = 1 - \prod_{j=1}^{n} (1 - w_j(x_j)), \\ &\quad w_j(z) = \begin{cases} 0.01 \cdot \exp(-(\frac{z}{20})^{2.5}) & \text{for} \quad j = 1, 2 \\ 0.01 \cdot \exp(-\frac{z}{15}) & \text{for} \quad j > 2 \end{cases} \end{aligned} \tag{3.3}$$

As in the previous example, we used a grid of 3×3, , 10 test points and 4 subdivision steps, we used $\epsilon = [2, 0.0004]^T$ which represents the accepted deterioration for each objective function. Figure 3.10 shows the result using pSCM. In this case pSCM is also able to find the set of approximate solutions.



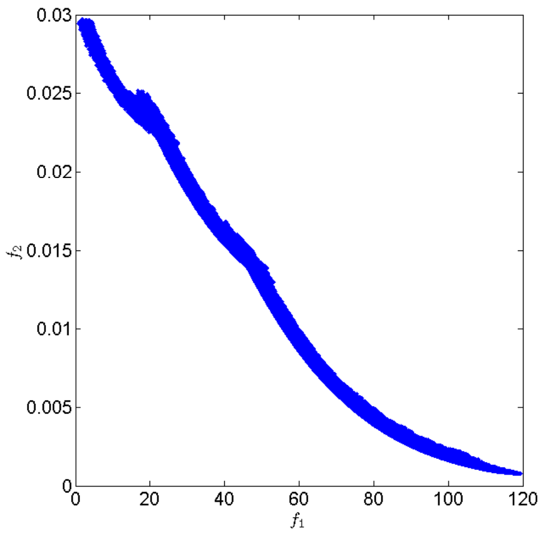


Figure 3.10: Approximate solutions of problem 3.3 with $\epsilon = [2, 0.0004]^T$. Approximate solutions are shown with red boxes.

Chapter 4

Parallel Simple Cell Mapping for Parametric Optimization Problems

One special kind of multi-objective optimization problems that has the same dimensionality problem are the PMOPs, The reason is that the total dimension is given by the number of dimensions of the problem n and the number of dimensions of the external parameter λ, the problem is $n + l$ dimensional in total, and its Pareto front is a $k - 1 + l$ manifold.

In this chapter we focus on the application of SCM for the treatment of PMOPs. On Section 4.1 we create our novel algorithm for the treatment of PMOPs,finally, some numerical results are on Section 4.1.1.

4.1 Parametric pSCM

We can use the idea of family of Pareto fronts and a family of Pareto sets to solve the PMOPs using pSCM algorithm, in this way, we can create a family of dynamical systems, which each decision variable is a λ_i space divided into a given $h_{\lambda i}$. The combination of each decision variable generates a l-dimensional hypercube, i.e. $\Lambda = \lambda_1 \times \ldots \times \lambda l$. For each vector $\lambda^{'} \in \Lambda$ we create a new function F. Finally, we solve the new problem using any SCM algorithm. In this way we propose the novel algorithm of Parametric pSCM (see Algorithm 12) described as follows:

Algorithm 12 Parametric pSCM

```
Require: The objective functions vector $F$, the external parameter $\lambda$, total cells $tot$
    in the system, the sizes $h$ per dimension and their limits $lim$, the depth $d$ of the
    search, and the maximal depth $md$.
Ensure: Approximation of the family of Pareto sets $Ps$ and the family of Pareto
    fronts $Pf$ of the PMOP.
 1: $\Lambda \leftarrow \emptyset$
 2: $Ps \leftarrow \emptyset$
 3: $Pf \leftarrow \emptyset$
 4: for all $\lambda_i \in \lambda$ do
 5:     $\Lambda_i \leftarrow$ Division of the $\lambda_i$ space in $h_{\lambda i}$
 6:     $\Lambda \leftarrow \Lambda \times \Lambda_i$
 7: end for
 8: for all $\lambda_j^{'} \in \Lambda$ do
 9:     $Ps_j, Pf_j \leftarrow$ pSCM($F_{\lambda_j^{'}}$,$tot$,$h$,$c_i.lim$,$d+1$,$md$)
10:     $Ps \leftarrow Ps \cup \lambda_j^{'} \times Ps_j$
11:     $Pf \leftarrow Ps \cup \lambda_j^{'} \times Pf_j$
12: end for
```

Algorithm 12 has two parts; the first part divides each parameter λ_i into $h_{\lambda i}$ pieces in order to create a cell division over the λ space. The second part searches for the global optima using each value of $\lambda_j^{'} \in \Lambda$, this task is perform using a unique function $F_{\lambda_j^{'}}$. After, we compute each individual problem using Algorithm 7, for each Pareto front and Pareto set obtained in this way, we perform a Cartesian product using the corresponding $\lambda_j^{'}$ vector, finally we store the resulting points into an archive. At the end of the second part we will have our approximation of the family of Pareto fronts.

4.1.1 Numerical Results

Here we present some numerical results of the novel Parametric pSCM algorithm. For the problems, we have chosen to take the bi-objective ones from book[42]. In all cases we have used for the dimension of the parameter space. For the computations, we have used an Alienware Mx17 R4 laptop with 8GB RAM and the following characteristics: (i) a Nvidia gtx 680m GPU with 1344 CUDA cores and a 720 MHz processor clock,

Table 4.1: Number of restarts and divisions per coordinate direction for the MOPs considered in this study.

	restarts	divisions per coordinate	divisions per lambda parameter
$PMOP1$	1	[32 32]	[30]
$PMOP2$	1	[32 32]	[30]
$PMOP3$	1	[32 32]	[30]
$PMOP4$	1	[32 32 32]	[30]
$PMOP5$	1	[32 32]	[30]

(ii) an Intel Core i7-3720QM CPU with a clock speed of 2.6 GHz and a maximal turbo frequency of 3.6 GHz. The design parameters of pSCM for each problem can be found in Table 4.1.

PMOP1

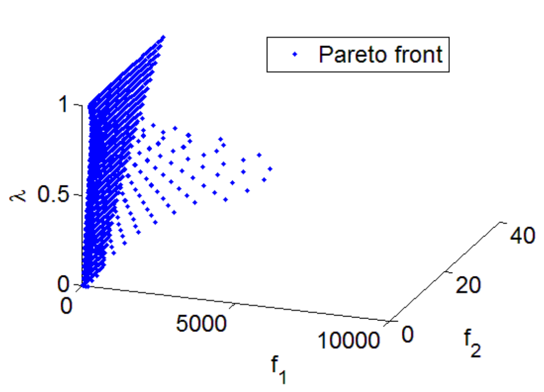
The following problem is highly non-linear in one objective and is defined as follows:

$$\begin{aligned} &\min F : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^2, F = (f_1, f_2) \\ &f_1(x,\lambda) = \lambda((x_1-2)^2 + (x_2-2)^2) + (1-\lambda)((x_1-2)^4 + (x_2-2)^8) \\ &f_2(x,\lambda) = (x_1+2\lambda)^2 + (x_2+2\lambda)^2 \end{aligned} \tag{4.1}$$

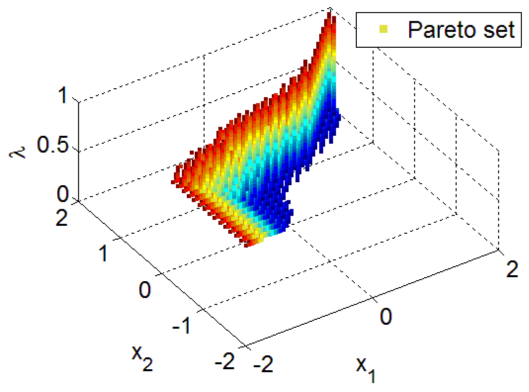
The result using $\Omega = [-2,2]^2$ and $\lambda = [0,1]$ is presented in Figure 4.1.

PMOP3

The following problem has different Pareto fronts given each λ (convex, concave or both):



(a) family of Pareto fronts



(b) family of Pareto sets

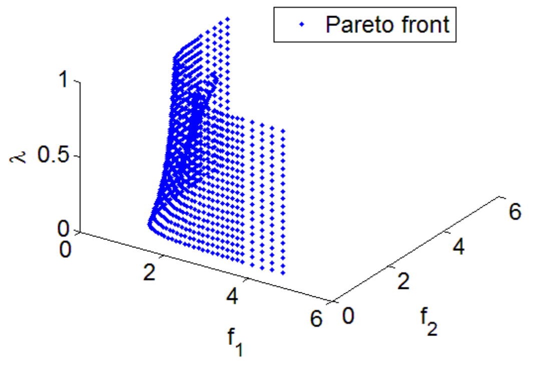
Figure 4.1: Result of PpSCM on PMOP1

PMOP2

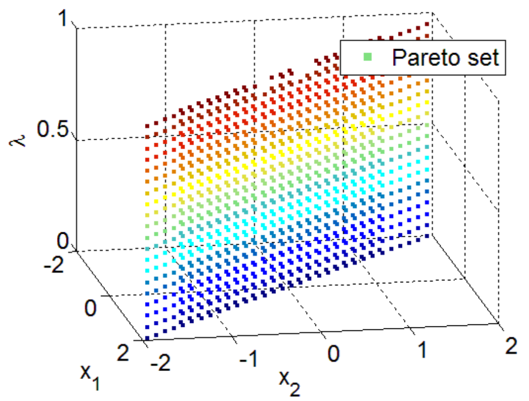
The following problem has different Pareto fronts given each lambda (convex, concave or both):

$$\begin{aligned}&\min F : \mathbb{R}^2 \times \mathbb{R} \rightarrow \mathbb{R}^2, F = (f_1, f_2)\\&f_1(x,\lambda) = 0.5(\sqrt{1+(x_1+x_2)^2} + \sqrt{1+(x_1-x_2)^2} + x_1 - x_2) + \lambda \exp^{-(x_1-x_2)^2}\\&f_2(x,\lambda) = 0.5(\sqrt{1+(x_1+x_2)^2} + \sqrt{1+(x_1-x_2)^2} + x_1 + x_2) + \lambda \exp^{-(x_1-x_2)^2}\end{aligned} \tag{4.2}$$

The result using $x = [-2, 2]^2$ and $\lambda = [0, 1]$ is presented in Figure 4.2.

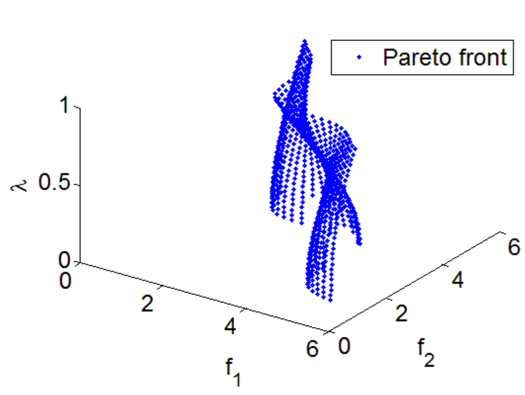


(a) family of Pareto fronts

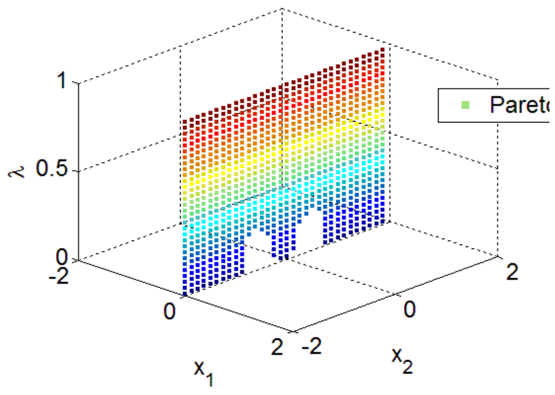


(b) family of Pareto sets

Figure 4.2: Result of PpSCM on PMOP2



(a) family of Pareto fronts



(b) family of Pareto sets

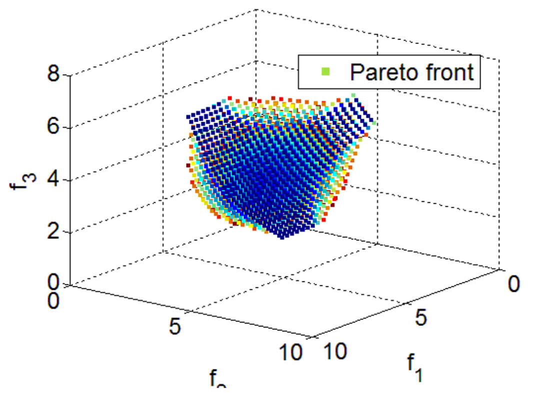
Figure 4.3: Result of PpSCM on PMOP3

$$
\begin{aligned}
&\min F : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^2, F = (f_1, f_2)\\
&f_1(x,\lambda) = \sqrt{1+(x_1^2)} + \sqrt{1+(x_2^2)} + \exp^{-(x_1-\lambda)^2} + \exp^{-(x_2+\sqrt{1+(x_2^2)}+\lambda)^2} - x_2\\
&f_2(x,\lambda) = \sqrt{1+(x_1^2)} + \sqrt{1+(x_2^2)} + \exp^{-(x_1-\lambda)^2} + \exp^{-(x_2+\lambda)^2} + x_2
\end{aligned} \tag{4.3}
$$

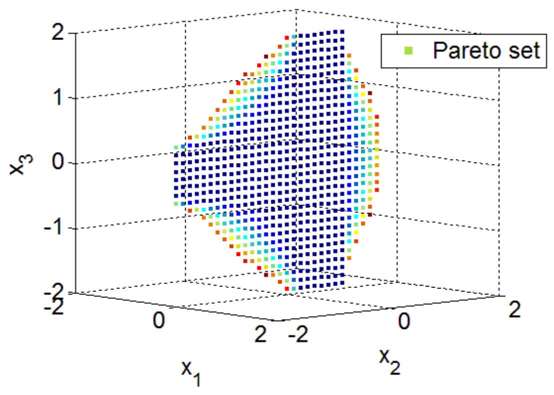
The result using $x = [-2,2]^2$ and $\lambda = [0,1]$ is presented in Figure 4.3.

PMOP4

The following problem has different Pareto fronts given each λ (convex, concave or both), also is three objective:



(a) family of Pareto fronts



(b) family of Pareto sets

Figure 4.4: Result of PpSCM on PMOP4

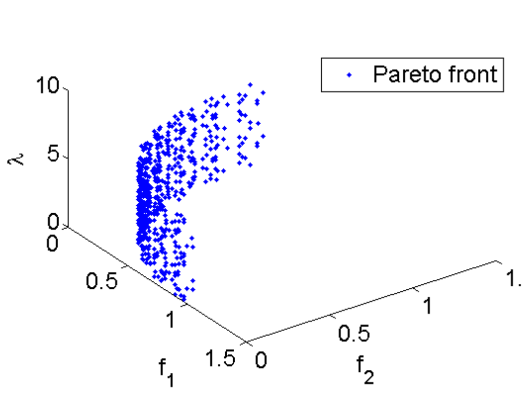
$$
\begin{aligned}
&\min F : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^3, F = (f_1, f_2, f_3)\\
&f_1(x,\lambda) = \sqrt{1+(x_1^2)} + \sqrt{1+(x_2^2)} + \sqrt{1+(x_3^2)} + \lambda \exp^{-(x_2^2 - x_3^2)} + \sqrt{2}x_2\\
&f_2(x,\lambda) = \sqrt{1+(x_1^2)} + \sqrt{1+(x_2^2)} + \sqrt{1+(x_3^2)} + \lambda \exp^{-(x_2^2 - x_3^2)} - .5\sqrt{2}x_2 + \sqrt{\frac{3}{2}}x_3\\
&f_3(x,\lambda) = \sqrt{1+x_1^2} + \sqrt{1+x_2^2} + \sqrt{1+x_3^2} + \lambda \exp^{-(x_2^2 - x_3^2)} - .5\sqrt{2}x_2 - \sqrt{\frac{3}{2}}x_3
\end{aligned}
\tag{4.4}
$$

The result using $x = [-2, 2]^3$ and $\lambda = [0, 1]$ is presented in Figure 4.4.

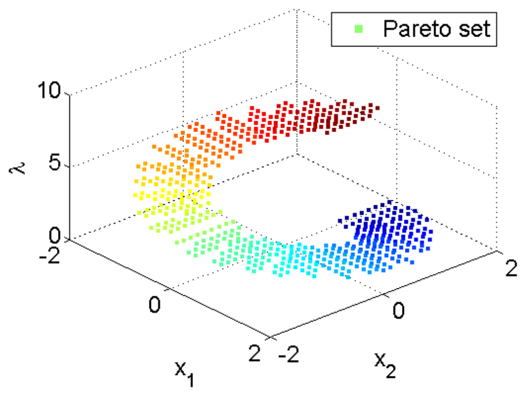
PMOP5

The following problem is a convex problem:

$$
\begin{aligned}
&\min F : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^2, F = (f_1, f_2)\\
&f_1(x,\lambda) = (x_1 - \sin(\lambda))^2 + (x_2 - \cos(\lambda))^2\\
&f_2(x,\lambda) = (x_1 - 2\sin(\lambda))^2 + (x_2 - 2\cos(\lambda))^2
\end{aligned}
\tag{4.5}
$$



(a) family of Pareto fronts



(b) family of Pareto sets

Figure 4.5: Result of PpSCM on PMOP5

The result using $x = [-2, 2]^2$ and $\lambda = [0, \pi]$ is presented in Figure 4.5.

Chapter 5

Parallel Simple Cell Mapping for Bilevel Optimization Problems

In this chapter we focus on the treatment of BMOPs. Section 5.1 we create our novel algorithm for the treatment of BMOPs, some numerical results of BSOPs are Section 5.2 and finally, in Section 5.3, we present some numerical results for different BMOPs.

5.1 Bilevel pSCM

As we mention in Chapter 2, the BMOP's are a special class of optimization problems, where the lower level task is a parametric optimization problem. Therefore we can use the same idea of PpSCM (see Algorithm. 12), it means to use a nested pSCM in order to solve the lower level optimization task, and control the search over the upper level using a dominance check over the upper objective function.

Since we have two simultaneous optimization proceses we have to select our priority task. In Table 5.1 we see that the most important task to find a bilevel optimum solution is to minimize any point in the lower objective decision vector and then search for the best values in the upper level objectives.

The BpSCM (see algorithm 13) is as follows: first we create a box over $x = x_u \cup x_l$ using the box constraints of the problem, then we divide each dimension of the box

Table 5.1: Quality of a candidate solution.

	Optimum in the lower level	Non-optimum in the lower level
Optimum in the upper level	The candidate is a Pareto point, the problem is a cooperative problem	The candidate is not feasible
Non-optimum in the upper level	The candidate is a feasible point, it could be a Pareto point	The candidate is not feasible

into *div* elements, after that, we perform pSCM on F_l returning a set s of non-dominated cells, then we perform a dominance check over F_u on s. Finally, we use those cells to repeat the process until md number of iterations are done.

Algorithm 13 BpSCM

Require: Total cells tot in the system, the sizes h per dimension and their limits lim, the depth d of the search, the maximal depth md, the upper objectives functions F_u,the lower objectives functions F_l.
Ensure: Approximation of the Pareto set and Pareto front of the MOP.

```
1: $cs \leftarrow \emptyset$
2: if $d = md$ then
3:     Store candidate set $cs$.
4: else
5:     $Ps_j, Pf_j \leftarrow$ pSCM($F_l$,$tot$,$h$,$c_i.lim$,$d+1$,$md$)
6:     $cs \leftarrow$ DominanceTest($tot$) over $F_u$
7:     for all $c_i \in cs$ do
8:         $h \leftarrow h/tot$
9:         BpSCM($F_u$,$F_l$,$tot$,$h$,$c_i.lim$,$d+1$,$md$)
10:    end for
11: end if
```

The BpSCM (see algorithm 13) is as follows: first we create a box over $x = x_u \cup x_l$ using the box constraints of the problem, then we divide each dimension of the box into *div* elements, after that, we perform pSCM on F_l returning a set s of non-dominated cells, then we perform a dominance check over F_u on s, finally, we use those cells to repeat the process until md number of iterations are done.

Table 5.2: Number of restarts and divisions per coordinate direction for the BSOPs considered in this study.

	restarts	divisions per coordinate x_l	divisions per coordinate x_u
$TP9$	10	[3 3]	[3 3]
$TP10$	10	[3 3]	[3 3]
$SMD1$	10	[3 3]	[3 3]
$SMD2$	10	[3 3]	[3 3]

5.2 Bilevel Single-Objective Optimization

Here we present some numerical results of the novel parallel cell mapping algorithm for the treatment of BSOPs.

We compare against BLEAQ [38] using the TP benchmark suite [38] and SMD benchmark suite [39]. In order to make a fair comparison we use the function evaluations and the quality of the approximation (for upper and lower objective function). In all cases we have used $n = 4$ for the dimension of the parameter space. For the computations, we have used an Alienware Mx17 R4 laptop with 8GB RAM and the following characteristics: (i) a Nvidia gtx 680m GPU with 1344 CUDA cores and a 720 MHz processor clock, (ii) an Intel Core i7-3720QM CPU with a clock speed of 2.6 GHz and a maximal turbo frequency of 3.6 GHz. The design parameters of pSCM for each problem can be found in Table 5.2.

5.2.1 TP9

This test problem, where the lower level problem is a convex optimization task and the upper level is convex with respect to upper level variables. The two levels cooperate with each other.

Table 5.3 presents that our method has the same approximation of the optimum but we have a significantly less lower level function evaluations.

Table 5.3: TP9 Comparison

Lower FE	Upper FE	F	error	f	error
BLEAQ					
11714	53	0.000058	0.000058	1	0
BpSCM					
1782	891	0.000025	0.000025	1	0

Table 5.4: TP10 Comparison

Lower FE	Upper FE	F	error	f	error
BLEAQ					
9600	79	0.000053	0.000053	1	0
BpSCM					
1782	891	0.000025	0.000025	1	0

5.2.2 TP10

This test problem has the same properties of TP9.

Table 5.4 shows that our method has the same approximation of the optimum but we have a significantly less lower level function evaluations.

5.2.3 SMD1

This test problem, where the lower level problem is a convex optimization task and the upper level is convex with respect to upper level variables. The two level cooperate with each other.

Table 5.5 shows that our method retrieve a bad solution due to it get stuck in a local minimum.

Table 5.5: SMD1 Comparison

Lower FE	Upper FE	F	error	f	error
BLEAQ					
12748	217	0.002381	0.002381	0.00009	0.00009
BpSCM					
13122	6561	0.69444444	0.69444444	0	0

Table 5.6: SMD2 Comparison

Lower FE	Upper FE	F	error	f	error
BLEAQ					
4066	167	-0.011068	0.011068	0.026473	0.026473
BpSCM					
1782	891	-0.096817	0.096817	0.096817	0.096817

5.2.4 SMD2

This test problem is similar to the SMD1. How ever, there is a conflict between the upper level and lower level optimization task. Since the two levels are in conflict, a non-optimum lower level solution may lead to a better upper level function value than the true optimum for the bilevel problem.

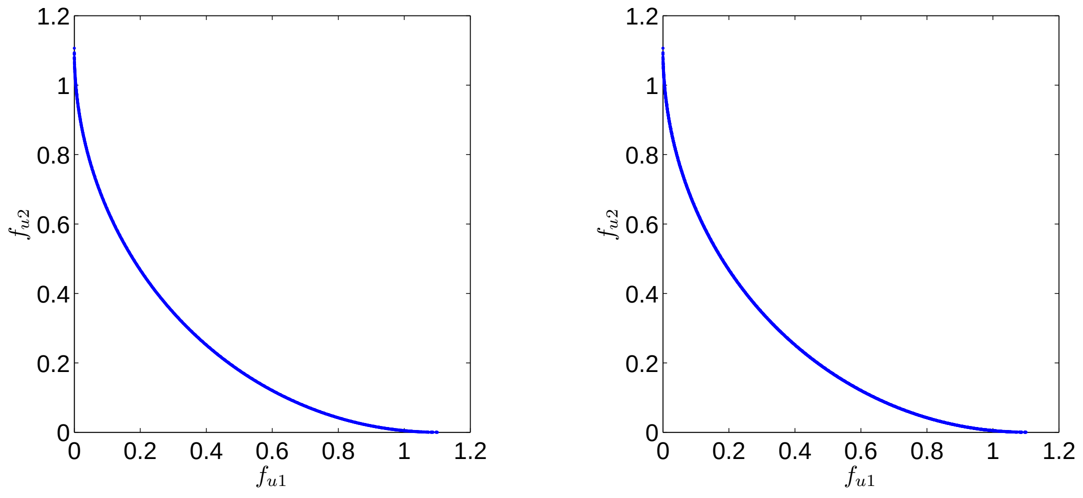
Table 5.6 presents that our method have an average performance due to it get stuck in a local minima, but it can handle problems with conflict between both levels.

5.3 Bilevel Multi-Objective Optimization

Here we present some numerical results of the BpSCM for the treatment of BMOPs.We use the DS-benchmark [8]. In order to make a fair comparison we us the function evaluations and Δ_2 of the approximation (for upper objective). For DS1 function we use $k = 1$ and $k = 5$ and for DS2 we use $k = 1$ and $k = 3$ (each problem are $2k$-dimensional). For the computations, we have used an Alienware Mx17 R4 laptop with 8GB RAM and the following characteristics: (i) a Nvidia gtx 680m GPU with 1344 CUDA cores and a 720 MHz processor clock, (ii) an Intel Core i7-3720QM CPU with a clock speed of 2.6 GHz and a maximal turbo frequency of 3.6 GHz. The design parameters of BpSCM for each problem can be found in Table 5.7.

Table 5.7: Number of restarts and divisions per coordinate direction for the BMOPs considered in this study.

	restarts	divisions per coordinate x_l	divisions per coordinate x_u
$DS1k = 1$	5	[3]	[3]
$DS1k = 5$	5	[3 3 3 3 3]	[3 3 3 3 3]
$DS2k = 1$	5	[3]	[3]
$DS2k = 3$	4	[3 3 3]	[3 3 3]



(a) DS1 $k = 1$ (b) DS1 $k = 5$

Figure 5.1: Results of BpSCM on DS1 problem.

5.3.1 DS1

This problem has $2k$ variables with k real-valued variables each for lower and upper levels. Lower level problem has multi-modalities. We use $r = 0.1, \alpha = 1, \gamma = 1$ and $\tau = 1$.

Figure 5.1 shows the final approximation of the BpSCM, we present in Table 5.8 and 5.9 the results of the approximation, we can see that BpSCM solves the problem, but even with the subdivision techniques, when the number of dimensions increases, the algorithm becomes too expensive.

Table 5.8: DS1 Comparison with $k = 1$

Lower FE	Upper FE	Upper level Δ_p
BpSCM		
52182	26091	0.00039997

Table 5.9: DS1 Comparison with $k = 5$

Lower FE	Upper FE	Upper level Δ_p
BpSCM		
684614106	342307053	0.00046061

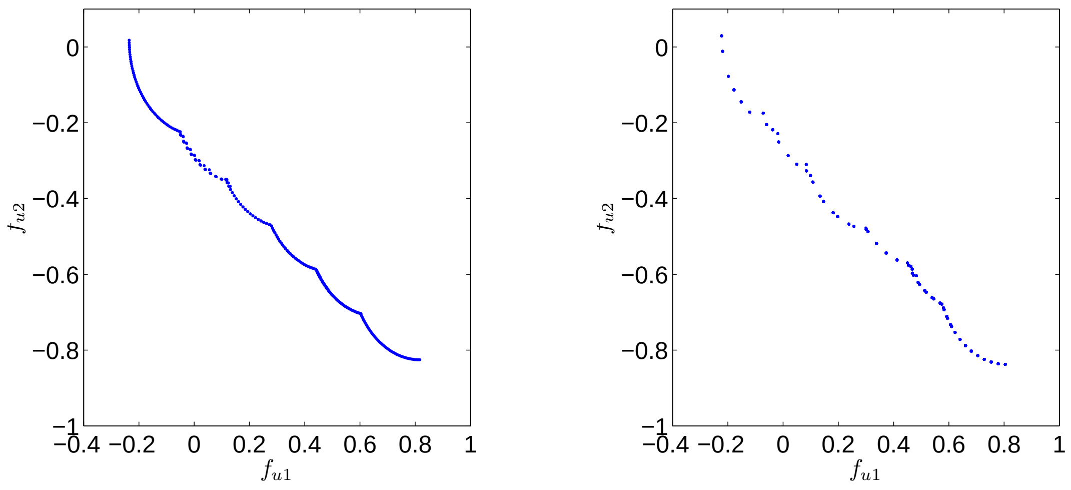
5.3.2 DS2

The next problem uses the ϕ_U parametric function, which causes a few discrete values of y_1 to determine the upper level Pareto optimal front.

Figure 5.2 shows the final approximation of the BpSCM, we present in Table 5.10 and 5.11 the results of the approximation, we can see that BpSCM solve the problem, but even with the subdivision techniques when we increase the number of dimensions the algorithm becomes too expensive, even more, in this problem in particular, each iteration of the algorithm return too many candidate cells.

Table 5.10: DS2 Comparison with $k = 1$

Lower FE	Upper FE	Upper level Δ_p
BpSCM		
659916	329958	0.0096



(a) DS2 $k = 1$ (b) DS2 $k = 5$

Figure 5.2: Results of BpSCM on DS2 problem.

Table 5.11: DS2 Comparison with $k = 3$

Lower FE	Upper FE	Upper level Δ_p
BpSCM		
178800858	89400429	0.0233

Chapter 6

Real World Application: Rotary Flexible Joint Module

Full state feedback control is an important part of the modern control theory [3]. Because of its vast applications in industries, there have been many studies to develop design or tuning techniques of the control. The well-known linear quadratic regulator (LQR) is the most popular optimal controller design in the modern control theory [2]. Since feedback controls are often designed to meet multiple and possibly conflicting performance goals, comprehensive studies are usually carried out to tune control gains in order to achieve the best overall performance [5, 41]. In the last three decades there have been a large number of publications on multi-objective optimal design of feedback controls, but few of those are related to bilevel optimization. In each level there is a multi-objective optimal control design that can be carried out in time domain or frequency domain. Time domain approach uses the time domain specifications of the closed-loop response as the objective functions such as overshoot, peak time, settling time and tracking error [21]. On the other hand, frequency domain design uses phase and gain margins as the objectives, and can consider robust issues such as model uncertainty, load disturbance and measure noise. Multi-objective optimization with robustness often involves the optimization among several norms. Vroemen and Jager reviewed the multi-objective design of robust controls for linear systems [40]. A

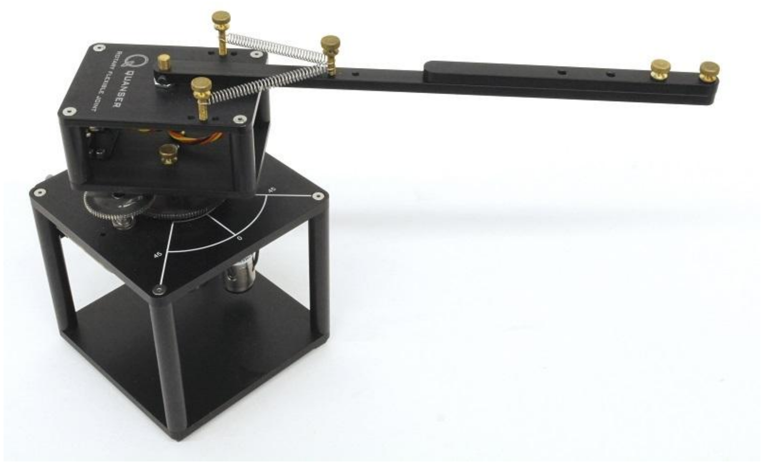


Figure 6.1: Rotary Flexible Joint module

normal Rotary Flexible Joint module (see Figure 6.1) consists of a free arm attached to two identical springs. The springs are mounted to an aluminum chassis which is fastened to the Rotary Servo Base Unit load gear. The module attaches to a DC motor on the Servo Base Unit that rotates a beam mounted on a flexible joint. The full state feedback tracking control of this module has been studied [28], in which the best tracking response of the system according to the time domain specifications.

If we want to suppress the vibration of the beam via a DC motor, the system can not be solved by a multi-objective optimization,instead, a bilevel formulation is possible.

6.1 The Design of the Problem

Feedback controls are often designed to achieve the best tracking response of the system according to the time domain specifications such as the peak time, settling time, overshoot and steady-state tracking error. The state vector describing the system dynamics is given as $x = (\theta, \alpha, \dot{\theta}, \dot{\alpha})$, where θ is the angle of the base and α is the relative angle of the flexible arm with respect to the base. The control goal is to let θ follow a given command while keeping the vibration of α minimal. the state equation for this system has the following form:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t), \tag{6.1}$$

where the state matrices are given by

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 628.5625 & -40.4033 & 0 \\ 0 & -1024.7473 & 40.4033 & 0 \end{bmatrix}, \mathbf{B} = [0, 0, 61.7567, -61.7567]^T. \tag{6.2}$$

To achieve tracking control, we need to augment the original dynamic system by introducing an additional state variable $x_i = \int_0^t (\theta - \theta_d)\, ds$. The state equation for the augmented system can be written as

$$\dot{\mathbf{x}}_e(t) = \mathbf{A}_e\mathbf{x}_e(t) + \mathbf{B}_e u(t) + \mathbf{G}_e\theta_d(t), \tag{6.3}$$

where

$\mathbf{x}_e(t) = \left[\mathbf{x}^T, x_i^T\right]$ is the augmented state vector.

$\mathbf{A}_e, \mathbf{B}_e$ and $\mathbf{G}_e$ are given as

$$\mathbf{A}_e = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{e} & 0 \end{bmatrix}, \mathbf{B}_e = \begin{bmatrix} \mathbf{B} \\ 0 \end{bmatrix}, \mathbf{G}_e = \begin{bmatrix} \mathbf{0} \\ -1 \end{bmatrix}, \tag{6.4}$$

where $\mathbf{e} = [1, 0, 0, 0]^T$.

The full state feedback control for the augmented system is given as

$$u(t) = -\mathbf{k}_e\mathbf{x}_e(t) + k_{p,\theta}\theta_d(t), \tag{6.5}$$

where

$$\mathbf{k}_e = -[k_{p,\theta}, k_{d,\theta}, k_{i,\theta}, k_{p,\alpha}, k_{d,\alpha}]. \tag{6.6}$$

The performance indices to be minimized are the overshoot M_p, settling time $t_{s,\theta}$

Table 6.1: Number of restarts and divisions per coordinate direction for the problem.

	restarts	divisions per coordinate x_l	divisions per coordinate x_u
$TP9$	7	[2 2 2]	[2 2 2 2]

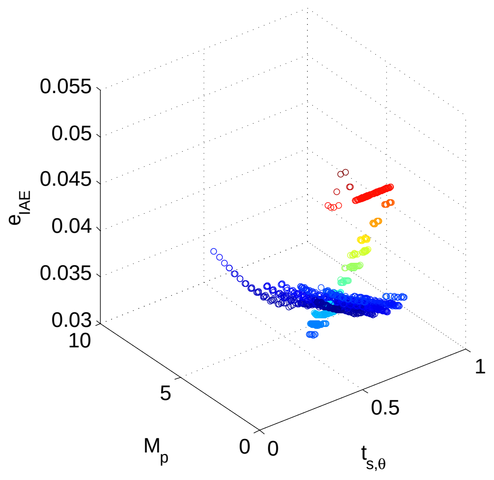
and absolute integrated error e_{IAE} of θ. The control objectives of θ are selected to optimize the tracking control as the upper level. In order to suppress the unwanted vibration of α, the maximum absolute response of α, denoted as $\max|\alpha|$, and the settling time of $t_{s,\alpha}$ are considered as the lower level. All theses objectives serve the control mission that θ follows the given command and α oscillates as little as possible. The final formulation of this control design can be stated as

$$\begin{aligned}&\min_{\theta,\alpha}\{M_p, t_{s,\theta}, e_{IAE}\}\\&\text{subject to}\\&\alpha \in \operatorname{argmin}\{\max|\alpha|, t_{s,\alpha}\}.\end{aligned} \tag{6.7}$$

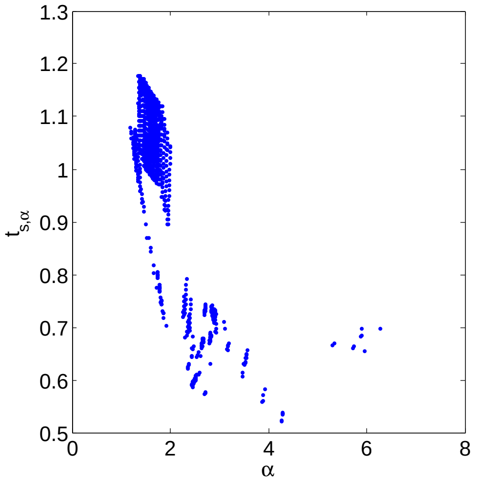
6.2 Numerical Results

Here we present the result of BpSCM, using the parameters shown in Table 6.1. For the computations, we have used an Alienware Mx17 R4 laptop with 8GB RAM and the following characteristics: (i) a Nvidia gtx 680m GPU with 1344 CUDA cores and a 720 MHz processor clock, (ii) an Intel Core i7-3720QM CPU with a clock speed of 2.6 GHz and a maximal turbo frequency of 3.6 GHz.

Figure 6.2 shows the upper and lower Pareto front obtained by BpSCM, the lower Pareto front contains the points of the families of the Pareto sets that are non-dominated in the upper level. In Figure 6.3 we can see the competitive nature of the problem since if we move faster to reach the desired displacement, it will take more settle time, and also the vibration of the beam will be easily controlled by the actuator, but in the other hand, if we want a soft movement, the actuator will work more making the beam unstable.

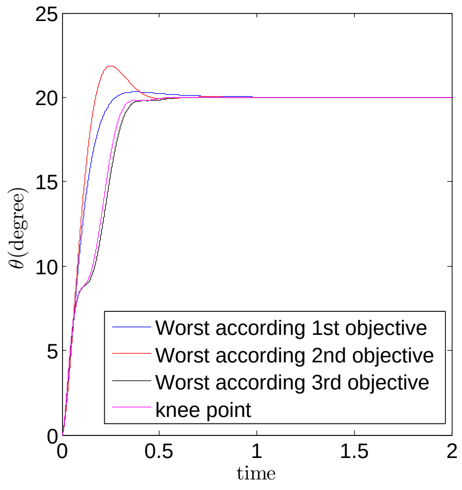


(a) UF

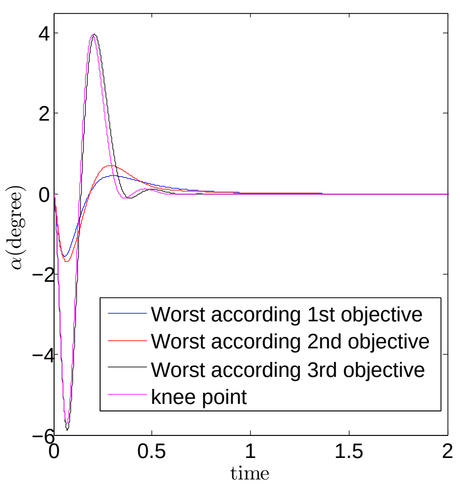


(b) BP3

Figure 6.2: Upper and lower Pareto front of the problem



(a) UF



(b) BP3

Figure 6.3: Numerical simulations of θ and α responses under Pareto optimal controls.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The main goal of this thesis has been to design set oriented methods for the numerical treatment of bilevel optimization problems. The proposed approach was to adapt the simple cell mapping method to this context. In Chapter 5 we have proposed BpSCM, which is a modification of pSCM algorithm, this algorithm is highly parallelizable. The numerical results show that BpSCM is competitive on low dimensional problems against a well known state of the art algorithm, and that a reasonable amount of function evaluations can be expected if the problem is small.

We have proposed parallel SCM in Chapter 3, it is a parallel implementation of the cell mapping technique for the approximation of the Pareto set/front of a given multi-objective optimization problem. Due to the data dependency of its two most costly tasks (creation of the cell space and evolution of the system), we have chosen a data parallelism approach. The implementation has been designed for the use of GPUs which allows the use of a large number of cores on a relatively small and cheap computer. Further, we proposed an upgrade of this algorithm, it is called pSCM algorithm. This algorithm uses subdivision techniques along archive techniques in order to reduce the number of cells inside of the cell space, in this way, larger problems can be solved.

We proposed PpSCM algorithm for the treatment of parametric optimization problems in Chapter 4, this algorithm can handle the problems proposed as test, but, the same dimensional problem appears when the dimensions of the external parameter increases, leading it to a work overflow on the CPU.

In Chapter 6 we give numerical evidence, which indicates that the BpSCM can be used to solve real world applications such as control design where a function evaluation take some seconds to be computed, the main reason is BpSCM is based on cell mapping techniques, and have its advantages like its global view of the problem and its highly parallelizable structure.

7.2 Future Work

One of the opportunities with the current algorithm is to improve the multimodality handling, since it is possible to find a candidate cell that dominates another candidate cell that contains the Pareto set, and it is possible that the algorithms get stuck in local minima points.

Further, it is expected that the constraint handling techniques have to be improved such as the additional consideration of equality constraints. Finally, we intend to apply the parallel algorithm to other problems such as constrained multi-objective optimization, and constrained bilevel optimization where the cell mapping ansatz will be advantageous.

In this thesis we proposed the use of cell mapping techniques along with subdivision techniques in order to handle more dimensions, but, it is possible to use different approaches to do this task, we propose, as a future work, the use of cell mapping techniques with evolutionary algorithms, in order to eliminate the dimensionality problems and still have the global view of the two approaches.

Bibliography

[1] Jonathan F. Bard. *Practical Bilevel Optimization: Algorithms and Applications.* Klugwer Academic Publishers, USA, 1st edition, 1999.

[2] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

[3] Zdzislaw Bubnicki. *Modern control theory*, volume 422. Springer, 2005.

[4] C. A. Coello Coello, G. B. Lamont, and D. A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems.* Springer, New York, second edition, September 2007. ISBN 978-0-387-33254-3.

[5] P Cominos and N Munro. PID controllers: Recent tuning methods and design to specification. *IEE Proceedings-Control Theory and Applications*, 149(1):46–53, 2002.

[6] L. G. Crespo and J. Q. Sun. Stochastic optimal control of nonlinear dynamic systems via Bellman's principle and cell mapping. *Automatica*, 39(12):2109–2114, Summer 2003.

[7] Kalyanmoy Deb and Ankur Sinha. Constructing test problems for bilevel evolutionary multi-objective optimization. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 1153–1160. IEEE, 2009.

[8] Kalyanmoy Deb and Ankur Sinha. An efficient and accurate solution methodology for bilevel multi-objective programming problems using a hybrid evolutionary-local-search algorithm. *Evolutionary computation*, 18(3):403–449, fall 2010.

[9] Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. *Scalable test problems for evolutionary multiobjective optimization.* Springer, USA, 1st edition, 2005.

[10] Michael Dellnitz, Oliver Schütze, and Thorsten Hestermeyer. Covering Pareto sets by multilevel subdivision techniques. *Journal of Optimization Theory and Applications*, 124(1):113–136, January 2005.

[11] Jesús Fernández Cruz, Oliver Schütze, Jian-Qiao Sun, and Fu-Rui Xiong. Parallel cell mapping for unconstrained multi-objective optimization problems. In Alexandru-Adrian Tantar, Emilia Tantar, Jian-Qiao Sun, Wei Zhang, Qian Ding, Oliver Schütze, Michael Emmerich, Pierrick Legrand, Pierre Del Moral, and Carlos A. Coello Coello, editors, *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V*, volume 288 of *Advances in Intelligent Systems and Computing*, pages 133–146. Springer International Publishing, 2014.

[12] Jesús Fernández Cruz, Oliver Schütze, Jian-Qiao Sun, and Fu-Rui Xiong. Simple cell mapping for multi-objective bi-level optimization problems. In Alexandru-Adrian Tantar, Emilia Tantar, Jian-Qiao Sun, Wei Zhang, Qian Ding, Oliver Schütze, Michael Emmerich, Pierrick Legrand, Pierre Del Moral, and Carlos A. Coello Coello, editors, *Extended Abstracts Collection—–EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V*, volume 288 of *Advances in Intelligent Systems and Computing.* Springer International Publishing, 2014.

[13] Jörg Fliege and Benar Fux Svaiter. Steepest descent methods for multicriteria optimization. *Mathematical Methods of Operations Research*, 51(3):479–494, 2000.

[14] Abhishek Gupta, Piaras Kelly, Matthias Ehrgott, and Simon Bickerton. Applying bi-level multi-objective evolutionary algorithms for optimizing composites manufacturing processes. In *Evolutionary Multi-Criterion Optimization*, pages 615–627. Springer, 2013.

[15] Carlos Hernández, Yousef Naranjani, Yousef Sardahi, Wei Liang, Oliver Schütze, and Jian-Qiao Sun. Simple cell mapping method for multi-objective optimal feedback control design. *International Journal of Dynamics and Control*, 1(3):231–238, 2013.

[16] Carlos Hernández, Jian-Qiao Sun, and Oliver Schütze. Computing the set of approximate solutions of a multi-objective optimization problem by means of cell mapping techniques. In Michael Emmerich, Andre Deutz, Oliver Schütze, Thomas Bäck, Emilia Tantar, Alexandru-Adrian Tantar, Pierre Del Moral, Pierrick Legrand, Pascal Bouvry, and Carlos A. Coello, editors, *EVOLVE – A Bridge between Probability, Set Oriented Numerics and Evolutionary Computation IV*, pages 171–188. Springer, 2013.

[17] Carlos Ignacio Hernández Castellanos. Cell-to-Cell mapping for Global Multi-Objective Optimization. Master's thesis, Departamento de Computación, CINVESTAV, México, 2003.

[18] C.S. Hsu. *Cell-to-cell mapping: A method of global analysis for nonlinear systems*. Applied mathematical sciences. Springer-Verlag, USA, 1st edition, 1987.

[19] Johannes Jahn. Multiobjective search algorithm with subdivision technique. *Computational Optimization and Applications*, 35(2):161–175, October 2006.

[20] Timoleon Kipouros. Stochastic optimisation in computational engineering design. In Oliver Schütze, Carlos A. Coello Coello, Alexandru-Adrian Tantar, Emilia Tantar, Pascal Bouvry, Pierre Del Moral, and Pierrick Legrand, editors, *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation II*, volume 175 of *Advances in Intelligent Systems and Computing*, pages 475–490. Springer Berlin Heidelberg, 2013.

[21] C Agees Kumar and N Kesavan Nair. Multi-objective PI controller design with an application to speed control of permanent magnet dc motor drives. In *Signal Processing, Communication, Computing and Networking Technologies (ICSCCN), 2011 International Conference on*, pages 424–429. IEEE, 2011.

[22] Rajeev Kumar and Peter Rockett. Improved sampling of the pareto-front in multiobjective genetic optimizations by steady-state evolution: a pareto converging genetic algorithm. *Evolutionary computation*, 10(3):283–314, 2002.

[23] Adriana Lara López. *Using Gradient Based Information to build Hybrid Multi-objective Evolutionary Algorithms.* PhD thesis, Departamento de Computacion, CINVESTAV, 2013.

[24] Yousef Naranjani, Carlos Hernández, Fu-Rui Xiong, Oliver Schütze, and Jian-Qiao Sun. A hybrid algorithm for the simple cell mapping method in multi-objective optimization. In M. Emmerich et al., editor, *EVOLVE – A Bridge between Probability, Set Oriented Numerics and Evolutionary Computation IV*, pages 207–223. Springer, 2013.

[25] Yousef Naranjani, Yousef Sardahi, Jesús Fernández Cruz, Oliver Schütze, Jian-Qiao Sun, and Fu-Rui Xiong. A simple cell mapping and genetic algorithm hybrid method for multi-objective optimization problems. In Alexandru-Adrian Tantar, Emilia Tantar, Jian-Qiao Sun, Wei Zhang, Qian Ding, Oliver Schütze, Michael Emmerich, Pierrick Legrand, Pierre Del Moral, and Carlos A. Coello Coello, editors, *Extended Abstracts Collection—–EVOLVE - A Bridge*

between Probability, Set Oriented Numerics, and Evolutionary Computation V, volume 288 of *Advances in Intelligent Systems and Computing*. Springer International Publishing, 2014.

[26] JohnM. Oliver, Timoleon Kipouros, and A.Mark Savill. A self-adaptive genetic algorithm applied to multi-objective optimization of an airfoil. In Michael Emmerich, Andre Deutz, Oliver Schütze, Thomas Bäck, Emilia Tantar, Alexandru-Adrian Tantar, Pierre Del Moral, Pierrick Legrand, Pascal Bouvry, and Carlos A. Coello, editors, *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation IV*, volume 227 of *Advances in Intelligent Systems and Computing*, pages 261–276. Springer International Publishing, 2013.

[27] Zhi-Chang Qin, Fu-Rui Xiong, Carlos Hernandez, Jesus Fernandez, Qian Ding, Oliver Schuetze, and Jian-Qiao Sun. Multi-objective optimal design of slide mode control with parallel simple cell mapping method, 2014.

[28] Zhi-Chang Qin, Shun Zhong, and Jian-Qiao Sun. Sliding mode control experiments of uncertain dynamical systems with time delay. *Communications in Nonlinear Science and Numerical Simulation*, 18(12):3558–3566, 2013.

[29] Thomas Rauber and Gudla Rünger. *Parallel Programming For Multicore and Cluster Systems*. Springer-Lehrbuch. Springer-Verlag, Germany, 2nd edition, 2007.

[30] Günter Rudolph, Boris Naujoks, and Mike Preuss. Capabilities of EMOA to detect and preserve equivalent Pareto subsets. In *Evolutionary Multi-Criterion Optimization*, pages 36–50. Springer, 2007.

[31] S. Ruuska and K. Miettinen. Constructing evolutionary algorithms for bilevel multiobjective optimization. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–7. IEEE, June 2012.

[32] Stefan Schäffler, Reinhart Schultz, and Klaus Weinzierl. Stochastic method for the solution of unconstrained vector optimization problems. *Journal of Optimization Theory and Applications*, 114(1):209–222, July 2002.

[33] Stefan Schäffler, Reinhart Schultz, and Klaus Weinzierl. Stochastic method for the solution of unconstrained vector optimization problems. *Journal of Optimization Theory and Applications*, 114(1):209–222, 2002.

[34] O Schütze, Xavier Esquivel, Adriana Lara, and Carlos A Coello Coello. Using the averaged Hausdorff distance as a performance measure in evolutionary multiobjective optimization. *Evolutionary Computation, IEEE Transactions on*, 16(4):504–522, August 2012.

[35] O Schütze, M. Laumanns, E. Tantar, C. A. Coello Coello, and E. Talbi. Computing gap free Pareto front approximations with stochastic search algorithms. *Evolutionary Computation*, 18(1):65–96, 2010.

[36] Oliver Schütze, Adriana Lara, and Carlos A. Coello. The directed search method for unconstrained multi-objective optimization problems. Technical Report COA-R1, CINVESTAV-IPN, 2010.

[37] Oliver Schütze, Massimiliano Vasile, Oliver Junge, Michael Dellnitz, and Dario Izzo. Designing optimal low-thrust gravity-assist trajectories using space pruning and a multi-objective approach. *Engineering Optimization*, 41(2):155–181, January 2009.

[38] Ankur Sinha, Pekka Malo, and Kalyanmoy Deb. Efficient evolutionary algorithm for single-objective bilevel optimization. *arXiv preprint arXiv:1303.3901*, abs/1303.3901, October 2013.

[39] Ankur Sinha, Pekka Malo, and Kalyanmoy Deb. Test problem construction for single-objective bilevel optimization. *Evolutionary computation*, June 2013.

[40] Bas Vroemen and Bram de Jager. Multiobjective control: An overview. In *Decision and Control, 1997., Proceedings of the 36th IEEE Conference on*, volume 1, pages 440–445. IEEE, 1997.

[41] Qing-Guo Wang, Tong-Heng Lee, Ho-Wang Fung, Qiang Bi, and Yu Zhang. PID tuning for improved performance. *Control Systems Technology, IEEE Transactions on*, 7(4):457–465, 1999.

[42] K. Witting. *Numerical Algorithms for the Treatment of Parametric Multiobjective Optimization Problems and Applications*. PhD thesis, Paderborn, Univ., Diss., 2012.

[43] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195, summer 2000.