Center of Research and Advanced Studies of the National Polytechnic Institute

Unit Zacatenco

Electrical Engineering Department
Computer Science Area

Use of Coevolution and Fitness Inheritance for Multi-Objective Particle Swarm Optimization

By:
María Margarita Reyes Sierra

as the fulfillment of the
requirement for the degree of:
Doctor of Science

Specialization in:
Electrical Engineering

Advisor:
Dr. Carlos Artemio Coello Coello

Mexico City, Mexico. August, 2006.

Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Ingeniería Eléctrica
Área de Computación

Uso de Coevolución y Herencia para Optimización Multi-Objetivo Mediante Cúmulos de Partículas

Tesis que presenta:
María Margarita Reyes Sierra

para obtener el grado de:
Doctora en Ciencias

En la especialidad de:
Ingeniera Eléctrica

Director de tesis:
Dr. Carlos Artemio Coello Coello

México D.F., México. Agosto de 2006.

To Efrén

To my parents

Acknowledgments

I thank my advisor, Dr. Carlos Coello, for his guidence and encouragement to reach this goal.

I thank my reviewers for their valuable comments to improve this work: Dr. Gerardo de la Fraga, Dr. Francisco Rodríguez, Dr. Carlos Mariano and Dr. Edgar Vallejo.

I thank my partners and friends: Goyito, Lalito, Nareli, Richard, Adriana, Santiago, Erika, Daniel, LuisVi y Vite. Thank you for the nice times.

I would like to thank Sofi for the love with which she always exceeds her dutys, being also the kindest friend.

I acknowledge support from CONACyT through a scholarship to pursue graduate studies at the Computer Science Section of the Electrical Engineering Department at CINVESTA-IPN, in México.

This research work was derived from CONACyT project “Advanced Multiobjective Optimization Techniques” (Ref. 45683-Y) whose Principal Researcher is Dr. Carlos A. Coello Coello.

Abstract

In this dissertation, we present our studies and proposals for reducing computational cost in multi-objective evolutionary optimization. First, we describe a coevolutionary multi-objective approach that we designed in the first stage of our work, whose main motivation was to achieve a computational cost reduction. The main idea of the proposed coevolutionary algorithm is to obtain information throughout the evolutionary process in order to subdivide the search space into subregions, use a subpopulation for each of these subregions and focus the search in the "promising" subregions of the search space. The proposed approach is validated using several test functions taken from the specialized literature. Then, with the aim of improving the results provided by the coevolutionary approach (especially in functions with high-dimensional decision search space), a new multi-objective particle swarm optimization algorithm was designed to replace the genetic algorithm originally adopted as search engine. This new multi-objective approach is based on the use of Pareto dominance, crowding factors, different mutation operators and ε-dominance. This approach is detailed and validated, and its parameters are studied by means of an analysis of variance. As a result, on-line mechanisms to adapt the values of the most important parameters are proposed. Also, several test functions are studied in order to explore those types of problems in which our algorithm has some difficulties. Furthermore, this algorithm is also tested with constrained functions and it is finally incorporated into the coevolutionary approach previously mentioned. After that, we provide an enhancement technique based on fitness inheritance, proposed to reduce the computational cost when applying the multi-objective particle swarm approach previously mentioned. In fitness inheritance, the fitness value of an offspring is obtained from the fitness values of its parents. In this way, we do not need to evaluate every individual at each generation, and the computational cost is reduced. We perform a study of several different techniques to incorporate fitness inheritance into our approach, and propose a dynamical scheme to obtain the larger possible amount of savings

(in terms of function evaluations), without affecting in a significant way the quality of the results. Finally, we discuss some theoretical issues related to the work developed, covering both particle swarm optimization and fitness inheritance.

Resumen

En esta Tesis, presentamos nuestros estudios y propuestas para reducir el costo computacional en optimización evolutiva multi-objetivo.

En primer lugar, se describe un enfoque coevolutivo multi-objetivo que fue diseñado durante la primera etapa de nuestro trabajo y cuya principal motivación fue reducir el costo computacional. La idea principal del algoritmo coevolutivo propuesto es obtener información a lo largo del proceso evolutivo para subdividir el espacio de búsqueda en subregiones y concentrar la búsqueda en las subregiones promisorias del espacio de búsqueda. La validación de este algoritmo se lleva a cabo usando varias funciones de prueba tomadas de la literatura especializada.

Posteriormente, se diseñó un nuevo algoritmo multi-objetivo basado en optimización mediante cúmulos de partículas, para sustituir el algoritmo genético que se estaba usando como motor de búsqueda en el algoritmo coevolutivo propuesto. Esto se hizo con el fin de mejorar el desempeño del algoritmo coevolutivo, especialmente en funciones con alta dimensionalidad en el espacio de las variables de decisión. Este nuevo algoritmo multi-objetivo está basado en el uso de dominancia de Pareto, factores de agrupamiento, diferentes operadores de mutación y dominancia ε. Así pues, se lleva a cabo la descripción y validación de este nuevo algoritmo, así como un estudio de sus parámetros por medio de un análisis de varianza. Como resultado, se proponen mecanismos en línea para adaptar los valores de los parámetros más importantes. Así también, se estudian varias funciones de prueba con el fin de explorar aquellos problemas que pudieran resultar difíciles para nuestro algoritmo. Además, el algoritmo se valida usando varias funciones de prueba restringidas, y finalmente se incorpora en el esquema coevolutivo antes mencionado.

Posteriormente, se propone una técnica de herencia para disminuir el costo computacional relacionado con el uso del algoritmo multi-objetivo basado en cúmulos de partículas, mencionado previamente. Cuando se usa herencia, no es necesario evaluar a todos los individuos en cada generación y el costo computa-

cional se reduce. A este respecto, se lleva a cabo un estudio de distintas técnicas para incorporar el concepto de herencia en nuestro algoritmo y se propone un esquema dinámico para obtener el mayor ahorro posible (en el número de evaluaciones).

Finalmente, se discuten algunos temas teóricos relacionados con el trabajo desarrollado, tanto en relación a la optimización mediante cúmulos de particulas como en relación a la herencia.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Due to the multicriteria nature of most real-world problems, multi-objective optimization problems are very common, particularly in engineering applications. As the name indicates, multi-objective optimization problems involve multiple objectives, which should be optimized simultaneously and that often are in conflict with each other. This results in a group of alternative solutions which must be considered equivalent in the absence of information concerning the relevance of the others.

Since Evolutionary Algorithms (EAs) deal with a group of candidate solutions, it seems natural to use them in multi-objective optimization problems to find a group of optimal solutions. Indeed, EAs have proved very efficient in solving multi-objective optimization problems [21, 25].

Motivation

In many real world problems, the evaluation of the objective function is usually very expensive (computationally speaking). In engineering design, for example, one objective function evaluation may take hours, days or even weeks. For this reason, in some cases the application of EAs is also very expensive, and becomes limited due to their population-based nature (which implies a great number of function evaluations). Despite the considerable volume of research on evolutionary multi-objective optimization [21], little emphasis has been placed on certain algorithmic design aspects such as efficiency [27, 57, 19]. Furthermore, research efforts to decrease the computational cost have been focused on EAs to solve single-objective optimization problems [53].

Objectives

Based on the hypothesis that it is possible to design approaches to decrease the computational cost of multi-objective evolutionary algorithms, the main motivation of the work reported here was precisely how to design such new mechanisms able to reduce computational cost, in terms of the number of function evaluations performed. More precisely, the main objective of this work was to develop mechanisms able to obtain results of the same quality of those corresponding to the approaches representative of the state of the art, but performing a smaller number of function evaluations.

Our Work

First, since the use of coevolutionary mechanisms (which have strong links to game theory [4]) has been scarce in the evolutionary multi-objective optimization literature, we decided to take advantage of some coevolutionary concepts to design a multi-objective evolutionary algorithm that could be more efficient (in terms of fitness function evaluations). With this aim, the proposed approach used information obtained throughout the evolutionary process to focus the search only on the promising sub-regions of the search space.

Since the ability of the coevolutionary approach proposed was limited when using it for solving multi-objective problems with high-dimensional decision variable spaces, we decided to replace the genetic algorithm, used as search engine, with a particle swarm optimization approach proposed by the author of this work.

Particle swarm optimization is a heuristic search technique that simulates the movements of a flock of birds which aim to find food. This heuristic has been found to be very effective in a wide variety of applications, being able to produce very good results at a very low computational cost [56, 32]. The success of the particle swarm optimization algorithm as a single-objective optimizer (mainly when dealing with continuous search spaces) and its relative simplicity have motivated researchers to extend the use of this bio-inspired technique to other areas. The fact that this is a population-based technique has made it a natural candidate to be extended for multi-objective optimization.

In this way, we proposed a new multi-objective particle swarm algorithm which was found to provide very competitive results. We also conducted studies about the parameters of the new approach, on-line adaptation mechanisms, possible difficulties with different types of test functions, and its ability to handle

constraints.

Finally, after testing the coevolutionary approach with the new particle swarm algorithm as its search engine and finding that the results obtained on high-dimensional test functions were not of the quality expected, we decided to implement different techniques to reduce computational cost. Thus, we proposed to incorporate the concept of fitness inheritance into our multi-objective particle swarm approach. In fitness inheritance, the fitness value of an offspring is obtained from the fitness values of its parents. In this way, we do not need to evaluate every individual at each generation, and the computational cost is reduced (by reducing the number of function evaluations performed). It is worth noting that the concept of fitness inheritance had never been used to reduce the computational cost of a real-coded particle swarm optimizer to solve multi-objective problems. As we will see, by using the concept of fitness inheritance, we were able to reduce the number of function evaluations by a 30%, without affecting the quality of the obtained results.

Outline

This document is organized in the following way:

- **Chapter 1**. Introduction (this chapter).
- **Chapter 2**. States the multi-objective problem and presents a brief introduction to EAs.
- **Chapter 3**. Describes the coevolutionary scheme proposed in the first stage of our work and provides its corresponding empirical validation.
- **Chapter 4**. Describes the multi-objective particle swarm optimization algorithm proposed and provides its corresponding empirical validation. It also presents the studies performed on the parameters of the algorithmm and some of its possible difficulties.
- **Chapter 5**. Presents an enhancement technique based on fitness inheritance that we proposed in order to reduce the computational cost of the particle swarm approach previously described.
- **Chapter 6**. Presents the study of some theoretical issues related to the work developed, about particle swarm optimization and fitness inheritance.

- **Chapter 7**. Provides our conclusions and future work.

Chapter 2

Preliminary Concepts

2.1 Statement of Multi-Objective Problems

We are interested in solving problems of the type:

$$\text{minimize } \mathbf{f}(\mathbf{x}) := [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})] \tag{2.1}$$

subject to:

$$g_i(\mathbf{x}) \geq 0 \quad i = 1, 2, \ldots, m \tag{2.2}$$

$$h_j(\mathbf{x}) = 0 \quad j = 1, 2, \ldots, p \tag{2.3}$$

where $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ is the vector of decision variables, $f_i : R^n \rightarrow R$, $i = 1, \ldots, k$ are the objective functions and $g_i, h_j : R^n \rightarrow R$, $i = 1, \ldots, m$, $j = 1, \ldots, p$ are the constraint functions of the problem (m inequality contraints g_i and p equality constraints h_j).

To describe the concept of optimality in which we are interested, we will introduce some definitions.

Definition 1. Given two vectors $\mathbf{x}, \mathbf{y} \in \mathbf{R}^k$, we say that $\mathbf{x} \leq \mathbf{y}$ if $x_i \leq y_i$ for $i = 1, \ldots, k$, and that $\mathbf{x}$ **dominates** $\mathbf{y}$ (denoted by $\mathbf{x} \prec \mathbf{y}$) if $\mathbf{x} \leq \mathbf{y}$ and $\mathbf{x} \neq \mathbf{y}$.

Figure 2.1 shows a particular case of the **dominance relation** in the presence of two objective functions.

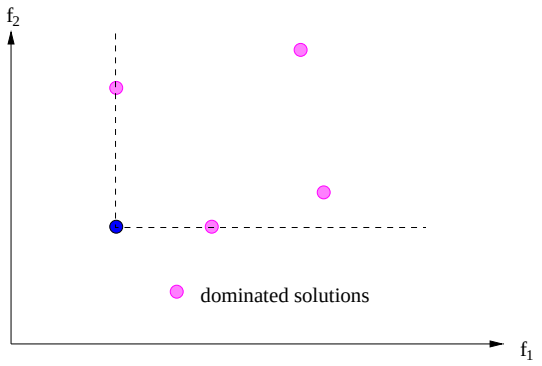


Figure 2.1: Dominance relation in a bi-objective space.

Definition 2. We say that a vector of decision variables $\mathbf{x} \in \mathcal{X}$, $\mathcal{X} \subset \mathbb{R}^n$, is **non-dominated** with respect to $\mathcal{X}$, if there does not exist another $\mathbf{x}' \in \mathcal{X}$ such that $\mathbf{f}(\mathbf{x}') \prec \mathbf{f}(\mathbf{x})$.

Definition 3. We say that a vector of decision variables $\mathbf{x}^* \in \mathcal{F}$, $\mathcal{F} \subset \mathbb{R}^n$ ($\mathcal{F}$ is the feasible region), is **Pareto-optimal** if it is nondominated with respect to $\mathcal{F}$.

Definition 4. The **Pareto Optimal Set** $\mathcal{P}^*$ is defined by:

$$\mathcal{P}^* = \{\mathbf{x} \in \mathcal{F} \,|\mathbf{x} \text{ is Pareto-optimal}\}$$

Definition 5. The **Pareto Front** $\mathcal{PF}^*$ is defined by:

$$\mathcal{PF}^* = \{\mathbf{f}(\mathbf{x}) \in \mathbb{R}^k | \mathbf{x} \in \mathcal{P}^*\}$$

Figure 2.2 shows a particular case of the **Pareto front** in the presence of two objective functions.
We thus wish to determine the Pareto optimal set from the set $\mathcal{F}$ of all the decision variable vectors that satisfy (2.2) and (2.3).

2.2 Mathematical Programming Techniques

There exist several techniques developed in Operations Research for solving multi-objective problems. These techniques can be classified considering the two main stages of the resolution process of a multi-objective problem:

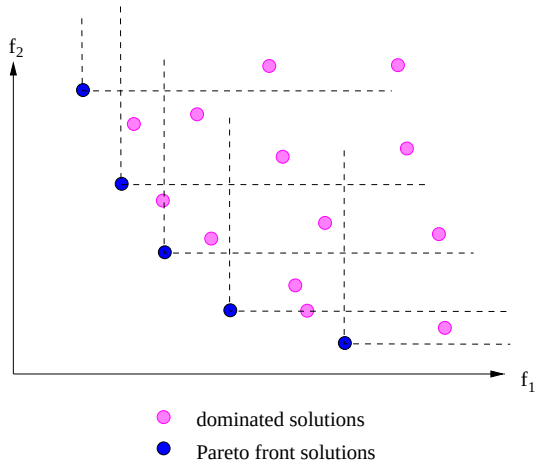


Figure 2.2: The Pareto front of a set of solutions in a two objective space.

1. Optimization of the objective functions.

2. Decision Making: Incorporation of preferences.

Given these two stages, mathematical programming techniques are usually classified into three main groups [22]:

- *A Priori* incorporation of preferences. Decision making takes place before searching. Examples: Goal Programming, Lexicographic method, Min-Max method, etc.

- *A Posteriori* incorporation of preferences. Decision making takes place after searching. Examples: ε-constraint method, linear combination of weights, etc.

- *Progressive* incorporation of preferences. Decision making and searching take place in an integrated form. Examples: STEP Method, Sequential Multi-objective Problem Solving Method, etc.

A disadvantage of these techniques is that sometimes they require information of the problem that is not always available. For example, some methods usually need the first or even the second derivative of the objective functions (and also the constraints). In those cases, if the objective function is not differentiable, those

methods can not be applied. On the other hand, most of these techniques usually have to be applied several times in order to obtain different solutions since they obtain just one solution each time [64]. For these reasons, alternative approaches are sometimes needed, for solving multi-objective optimization problems.

2.3 Evolutionary Algorithms

Evolutionary Algorithms (EA) are stochastic approaches used for solving optimization problems. The population-based feature of EAs makes them suitable for solving multi-objective optimization problems since such characteristic allows the generation of several elements of the true Pareto front in a single run.

Also, EAs have several advantages such as that they don't need any specific knowledge about the problem and that they are less susceptible to get trapped in local optima.

EAs are inspired on the "Neo-Darwinian" theory that establishes that all the richness of *life* in the planet can be explained by means of four processes:

- Reproduction. Mechanisms by which new individuals can be obtained by combining existing individuals.
- Mutation. Mechanism by which one new individual can be obtained by "changing" an existing individual.
- Competition. Different species sharing resources develelop competent behavior.
- Selection. Only the best individuals are able to survive.

In this way, in EAs a set of solutions (known as population) evolves through generations by means of the *natural selection* mechanism and the *survival of the fittest* principle. For such evolving process, the main components of an EA are:

- A specific encoding for the solutions of the problem (known as individuals). In EAs, individuals are represented by *chromosomes*. Chromosomes are data structures that encode the parameters of a possible solution of the problem. Also, chromosomes are usually represented by binary strings, but it is possible to use an integer or real encoding. See Figure 2.3.

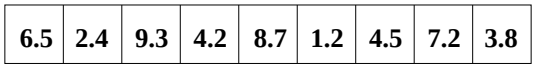


Figure 2.3: A chromosome with real encoding.

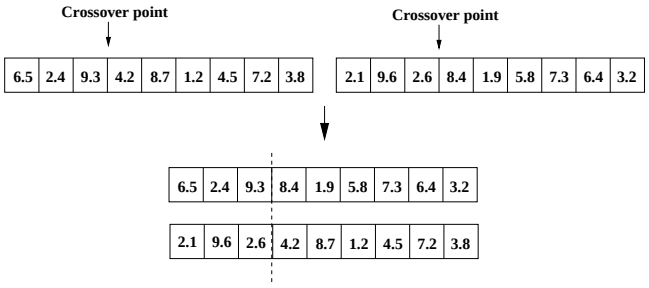


Figure 2.4: Crossover operator.

- Variation operators. EAs usually apply recombination and/or mutation operators in order to obtain new individuals. The recombination or crossover operator combines the chromosomes of the parents in order to obtain a new individual. There exist several types of crossover operators: of one point, two points, *n*-points, uniform, etc. In Figure 2.4, an example of the one point crossover operator is shown. The mutation operator obtains a new individual "changing" the value of one (maybe more) element of the original chromosome. There are also different mutation operators: uniform, non-uniform, etc. Figure 2.5 shows an example of the uniform mutation operator.

- A fitness function which plays the role of the environment. The fitness value

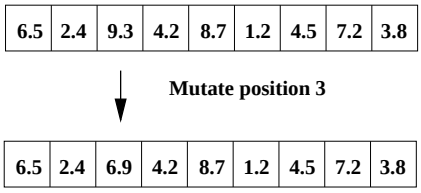


Figure 2.5: Mutation operator.

of an individual indicates "how good" is an individual with respect to others. In some cases, the fitness function is the objective function itself.

- A selection procedure which plays the role of the environmental pressure. Different selection techniques have been proposed for EAs [40]. These techniques are usually either stochastic or based on a tournament, both types of techniques are based on the fitness value of the individuals.

Based on different instances of these components, EAs can be classified into three main groups: Evolution Strategies, Evolutionary Programming and Genetic Algorithms. However, a paradigm recently proposed, called Swarm Intelligence, has also been considered a new type of EA. Such paradigms are described below.

2.3.1 Evolution Strategies

Evolution Strategies (ES) were developed by Peter Bienert, Ingo Rechenberg and Hans-Paul Schwefel [2] in the 60s, inspired in the mutation mechanism that exists in nature.

The original version of the ES was called (1+1)-ES. This ES has just one parent and creates just one child from this parent (by means of the mutation operator). The best individual between the parent and child becomes the new parent.

The mutation operator is defined in the following way: $\mathbf{x}' = \mathbf{x} + N(0,\sigma)$ where $N(0,\sigma)$ represents a random variable normally distributed with mean 0 and standard deviation σ. Usually, $\sigma = 1$.

In general, the population of an ES can have μ parent individuals and create λ offspring. Recombination in ES is possible but it is a secondary operator and the selection mechanisms adopted are normally deterministic. Schwefel proposed a notation for two different selection mechanisms: the $(\mu+\lambda)$-selection and the (μ,λ)-selection. In the former, the parents for the next generation are selected from the union of the μ parents and the λ offspring. In the second case, the parents for the next generation are selected only from the λ offspring. Figure 2.6 presents the general algorithm for the ES.

2.3.2 Evolutionary Programming

Lawrence J. Fogel developed Evolutionary Programming (EP) in 1964 [35]. With this technique, Fogel aimed to model the evolution of species. Since different species don't reproduce among themselves, EP has no recombination operator.

Begin
$t \leftarrow 0$
Initialize $P(t)$
Evaluate $P(t)$
Repeat
Apply recombination to $P(t)$ to generate $P'(t)$
Apply mutation to $P'(t)$ to generate $P''(t)$
Evaluate $P''(t)$
Select the best individuals to obtain $P(t+1)$ from:
$P(t) \cup P''(t)$ $(\mu+\lambda)$-selection
$P''(t)$ (μ,λ)-selection
$t \leftarrow t+1$
Until stop condition is reached
Report best solution in $P(t)$
End

Figure 2.6: General algorithm for ES.

The mutation operator used in EP is Gaussian and the selection mechanism is a $(\mu+\lambda)$-selection by means of stochastic tournaments. Figure 2.7 presents the general algorithm for EP.

2.3.3 Genetic Algorithms

Genetic Algorithms (GAs) are probably the most popular type of EA. While the biologist Fraser was the first to model biological systems by means of a computer simulation that can be considered an early GA [38], Holland is normally credited with the conception of this technique which he used to tackle machine learning tasks [45].

Recombination and mutation operators are both used in GAs. However, in GAs the recombination operator is the most important one and mutation is a secondary operator. Variation operators (recombination and mutation) are applied with a certain probability and the selection mechanism is stochastic (as in EP). Figure 2.8 presents the general algorithm for the GA.

```
Begin
    t ← 0
    Initialize P(t)
    Evaluate P(t)
    Repeat
            Apply mutation to P(t) to generate P'(t)
            Evaluate P'(t)
            Select P(t+1) from P(t) ∪ P'(t)
                by means of stochastic tournaments
            t ← t+1
    Until stop condition is reached
    Report best solution in P(t)
End
```

Figure 2.7: General algorithm for EP.

```
Begin
    t ← 0
    Initialize P(t)
    Evaluate P(t)
    Repeat
            Apply recombination to P(t) to generate P'(t)
            Apply mutation to P'(t) to generate P''(t)
            Evaluate P''(t)
            Select P(t+1) from P''(t)
                by means of stochastic tournaments
            t ← t+1
    Until stop condition is reached
    Report best solution in P(t)
End
```

Figure 2.8: General algorithm for GA.

2.3.4 Swarm Intelligence

Swarm Intelligence is related to the study of colonies, or swarms of social organisms. Studies of the social behavior of organisms in swarms inspired the design of efficient optimization algorithms. For example, the simulation of the choreography of bird flocks led to the design of the particle swarm optimization algorithm, and the studies of the foraging behavior of ants resulted in the ant colony optimization algorithm [31].

In these algorithms, individuals behave according to their past experience and the interaction with other individuals. These interactions usually cause a global behavior.

Since in our work we developed a particle swarm optimization algorithm, a more detailed description of this technique is provided in Chapter 4.

2.4 Multi-Objective Evolutionary Algorithms

The first Multi-Objective Evolutionary Algorithm (MOEA), called *Vector Evaluated Genetic Algorithm* (VEGA), was implemented by Schaffer in the mid-80s [92, 93]. VEGA basically consisted of a simple GA with a modified selection mechanism. At each generation, individuals were selected according to each one of the objective functions, in order to generate k sub-populations for a problem with k objective functions. Later, all sub-populations were mixed and the crossover and mutation operators were applied in order to obtain a new whole population. The main disadvantage of VEGA was that it suffered from the problem of "speciation", that is, the existence of "species" within the population, formed by individuals which were very good only on one of the objective functions but that were no good compromise solutions. This problem was due to the selection mechanism which selected individuals based only on one objective function, giving to compromise solutions few possibilities to survive.

The MOEAs that followed VEGA until the mid-90s used aggregating functions (usually linear) [51, 111], lexicographic ordering [37] and target-vector approaches [42].

2.4.1 First generation of MOEAs

The first generation of MOEAs arises from the proposal of David Goldberg in 1989, after analyzing VEGA, of using a selection mechanism based on the con-

cept of Pareto optimality [39]. Also, Goldberg suggested the use of a mechanism to maintain diversity within the population in order to prevent premature convergence, that is, the fast convergence of the population to a unique solution.

The most representative algorithms of this generation are the following:

1. **Nondominated Sorting Genetic Algorithm** (NSGA)
   This algorithm was proposed by Srinivas and Deb in 1994 [100]. The main characteristic of this algorithm is that before performing selection, the individuals are ranked in different levels in the following way: all nondominated individuals are classified into one category with rank 1 and receive a dummy fitness that allows them to have the same probability of being selected. Later, fitness sharing is applied on the individuals just classified in such a way that their fitnesses are penalized based on the number of individuals that share the same neighborhood (these neighborhoods are called *niches*, whose size is controlled through a parameter called *niche radius* (σ_{share}), which is defined by the user). Then, this group of individuals with rank 1 is ignored and the process is repeated. This time, the nondominated individuals will have rank 2 and a dummy fitness lower than that of the individuals with rank 1. The process continues until all individuals are classified. Figure 2.9 shows the ranking process of NSGA.

   Since individuals with rank 1 have the maximum fitness value, they will always be selected more times than the rest of the population. This behavior allows the search of nondominated individuals. On the other hand, the mechanism of fitness sharing helps to distribute the population along the Pareto front of the problem.

2. **Niched-Pareto Genetic Algorithm** (NPGA)
   This algorithm was proposed by Horn and Nafpliotis in 1994 [46]. The NPGA uses a tournament selection mechanism based on Pareto dominance: two individuals of the population are randomly chosen and compared against a subset of the population also randomly chosen (usually, around 10% of the population). If one of the individuals is dominated (by the subset of the population) and the other is not, the nondominated individual wins. If both individuals are either dominated or nondominated, the winner is chosen according to a mechanism based on fitness sharing. Figure 2.10 illustrates the niches of both individuals, used for the fitness sharing mechanism. It is worth noting that the NPGA does not rank the population.

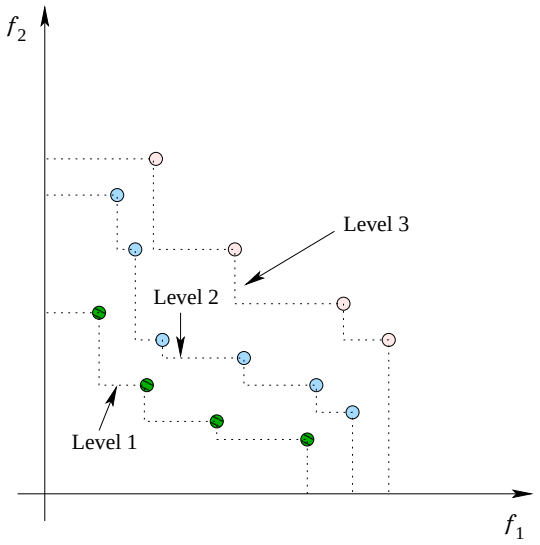


Figure 2.9: Ranking process of NSGA.

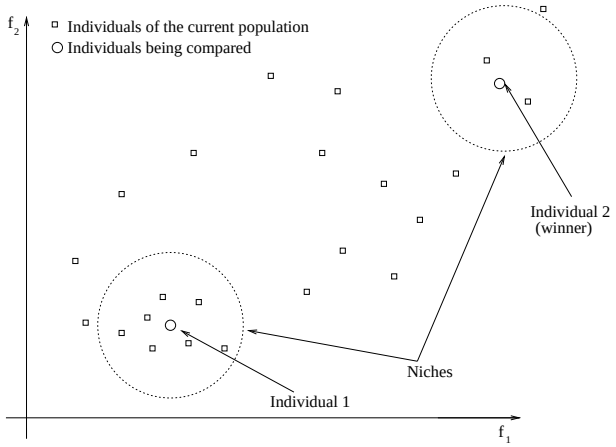


Figure 2.10: Mechanism of fitness sharing applied by the NPGA. The individual whose niche is less crowded, wins.

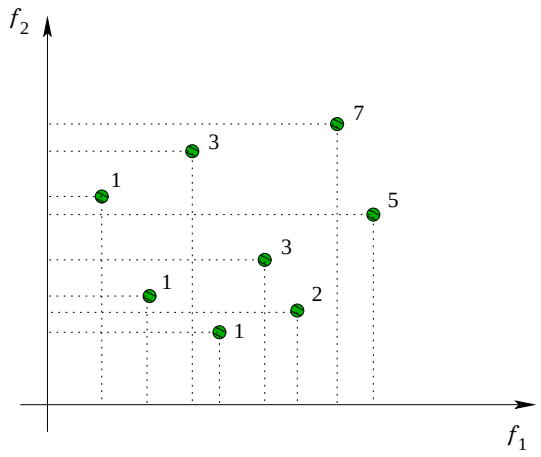


Figure 2.11: Ranking process of MOGA. The rank of each individual depends of the number of individuals that dominate it.

3. **Multi-Objective Genetic Algorithm** (MOGA).
   This algorithm was proposed by Fonseca and Fleming in 1993 [36]. In this algorithm, the rank of an individual is proportional to the number of individuals that dominate it. For example, the rank of the individual x_i at generation t, which is dominated by dom_i^t individuals is:

$$\texttt{rank}(x_i,t) = 1 + \text{dom}_i^t$$

   In this way, the nondominated individuals have rank 1 and the rest of the individuals are penalized according to the number of individuals that dominate them. Figure 2.11 shows the ranking process of MOGA.

   On the other hand, the fitness assignment is performed in the following way:

   - The population is sorted according to the rank of individuals.
   - Fitness of individuals is assigned by means of an interpolation process from the best to the worst individuals according to a function that in most cases is linear, but not necessarily.
   - Fitness of individuals with the same rank is averaged.
   - Finally, fitness sharing is applied (as in the case of NSGA):
     (a) First we calculate:

$$\phi(d_{ij}) = \begin{cases} 1 - (\frac{d_{ij}}{\sigma_{\text{share}}}), & d_{ij} < \sigma_{\text{share}} \\ 0, & \text{otherwise} \end{cases}$$

where d_{ij} is the euclidean distance (in the decision or objective function space) between individuals i and j, and σ_{share} is the niche radius.

(b) Fitness of individuals is modified in the following way:

$$f_{S_i} = \frac{f_i}{\sum_{j=1}^{M} \phi(d_{ij})}$$

where M is the size of the population.

This algorithm also implemented mating restrictions in such a way that individuals belonging to different niches can not mate.

The MOEAs of the first generation were characterized by the use of selection mechanisms based on Pareto dominance and the use of different fitness sharing mechanisms in order to maintain diversity within the population.

2.4.2 Second generation of MOEAs

The second generation of MOEAs took place with the introduction of the notion of elitism. In evolutionary multi-objective optimization, elitism is usually implemented through an external archive, also called secondary population, which stores the nondominated individuals found along the search. However, elitism can also be implemented through the use of $(\mu + \lambda)$-selection, by which, at each generation, parents and children are compared in order to select the best of them, to conform the next population. Nevertheless, in both cases of elitism, extra mechanisms are usually needed in order to obtain a well distributed final set of solutions.

In the following, we present the most representative algorithms of this generation:

1. **Strength Pareto Evolutionary Algorithm** (SPEA)
   This algorithm, proposed by Zitzler and Thiele [102], uses an external archive to store the nondominated individuals found along the search. For each one of the individuals stored in the external archive, this algorithm calculates a *strength* value similar to the rank assigned by MOGA. The strength value of an individual is proportional to the number of individuals that it dominates. The fitness of an individual is calculated according to the strength values of the individuals in the external archive, that dominate it. Also, SPEA uses a clustering technique, called *average linkage method* [66], to maintain diversity.

2. **Strength Pareto Evolutionary Algorithm 2** (SPEA2)
   The second version of SPEA was proposed by Zitzler et al. [117]. This new version has three main differences with respect to the old one: it assigns the fitness of an individual taking into account the number of individuals that it dominates, but also the number of individuals that dominate it; also, it uses a nearest neighbor density estimation technique that guides the search more efficiently; and it incorporates a mechanism for filtering the final set of solutions, that guarantees the preservation of boundary solutions.

3. **Pareto Archived Evolution Strategy** (PAES)
   This is perhaps the simplest MOEA possible and it was proposed by Knowles and Corne [57]. PAES consists of an (1+1)-ES (one parent generates one child) with an external archive, which stores some nondominated solutions found along the search. The most important aspect of this algorithm is the mechanism used to maintain diversity: an adaptive grid that sub-divides the objective space. Each individual is placed on a grid location that depends on its objective function values (which are used as coordinates). In this way, this algorithm maintains diversity by controlling the number of individuals placed on each grid location.

4. **Nondominated Sorting Genetic Algorithm II** (NSGA-II)
   The revised version of NSGA, called NSGA-II, was proposed by Deb et al. [26, 27]. This new version of the NSGA algorithm is more efficient (computationally speaking). The NSGA-II uses a *crowding* operator to maintain diversity, by preferring individuals that belong to the less crowded regions of the search space. Also, this algorithm introduces elitism by means of the $(\mu+\lambda)$-selection.

5. **Niched-Pareto Genetic Algorithm 2** (NPGA2)
   Erickson et al. proposed a revised version of the NPGA, called NPGA 2 [33]. The NPGA 2 uses Pareto ranking, but keeps the tournament selection mechanism previously described. Elitism was introduced in this algorithm by means of $(\mu+\lambda)$-selection, as in the case of the NSGA-II. In this case, niche counts are performed using the population that is being generated instead of the current population.

6. **Pareto Enveloped-based Selection Algorithm** (PESA)
   This algorithm was proposed by Corne et al. [24]. PESA is very similar to PAES, since the external population is larger than the main population

of the algorithm. Also, PESA uses the same adaptive grid to store the nondominated solutions. However, in this case, the external population also determines the selection method used by the algorithm (based on the same crowding measure used to determine which individuals are stored in the grid). There is also a revised version of PESA, called PESA-II, proposed by Corne et al. [23]. The main difference between PESA and PESA-II is that, in this case, selection is based on regions rather than individuals. The main motivation for this modification was to reduce computational cost associated with Pareto ranking.

7. **Micro Genetic Algorithm** (micro-GA)
   This approach, proposed by Coello and Toscano [18], consists of a genetic algorithm with a very small population (four individuals) combined with a reinicialization process and an external archive. Figure 2.12 shows how this algorithm works.

   First, a random population is generated. This population enters into the memory, which is divided in two parts: the replaceable and the nonreplaceable. The nonreplaceable part doesn't change throughout the evolutionary process and works as a source of diversity. On the other hand, the replaceable part changes at every cycle of the micro-GA. At every cycle of the micro-GA, the population of the genetic algorithm is taken (with certain probability) from both parts of the memory. In this way, within the population of the GA there are individuals randomly generated (nonreplaceable part) and individuals obtained after an evolutionary cycle (replaceable part). During each cycle, the crossover and mutation operator are applied in an usual way. After a complete cycle, two nondominated vectors are chosen from the final population and compared against the individuals stored in the external archive. If they (one or both) are nondominated, they are stored in the external archive.

   In this way, the micro-GA uses three forms of elitism: an external archive, a replaceable memory and a mechanism by which the population is partially filled with the best solutions found so far.

2.4.3 Test Functions

In this section, we provide the definitions of the test functions that are going to be used to validate the different approaches proposed as part of this work.

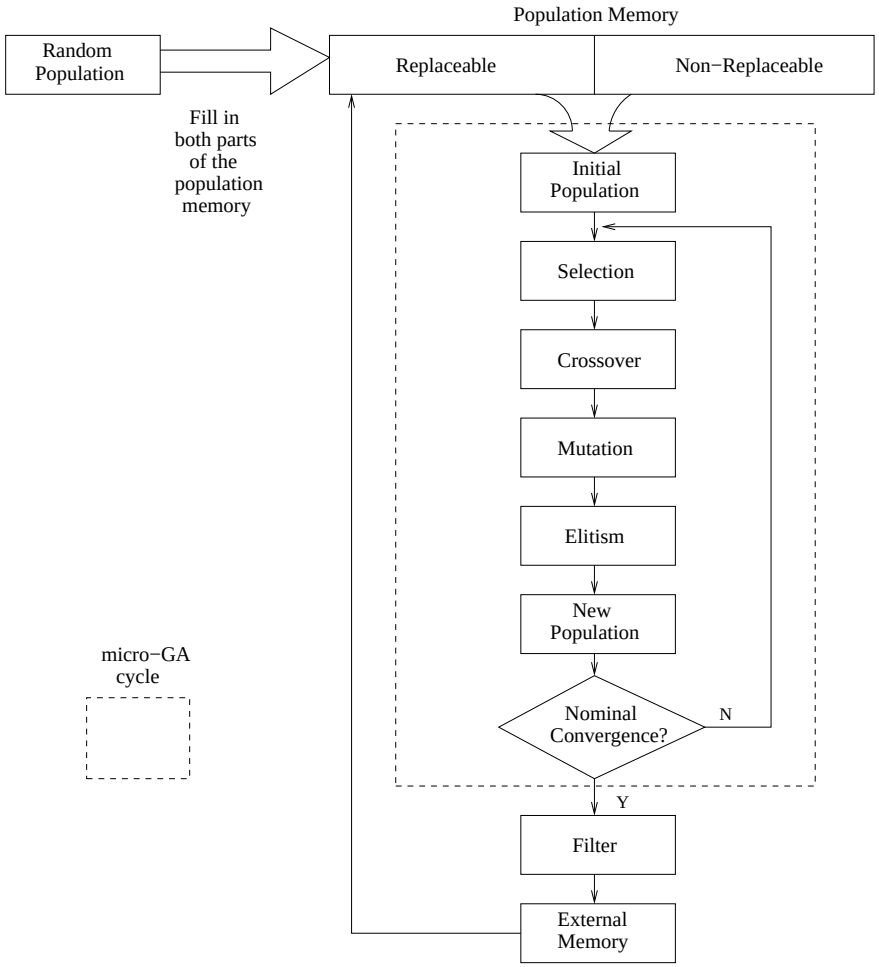


Figure 2.12: Diagram of the micro-GA.

It is important to note that these functions are representative of the benchmarks commonly used in the multi-objective optimization literature, to test new MOEAs. However, they don't necessarily reflect the characteristics of the optimization problems found in the real world. Nevertheless, some of these functions possess features that make them extremely difficult problems, even more than the problems that we could expect to find in real world applications. For this reason, when using these test functions we are hoping that if a new MOEA is able to solve them (that is, either to find Pareto optimal solutions or good approximations of them), it will be also able to solve real-world applications.

- Test Function Deb [21]:

  Minimize

  $$f_1(x_1,x_2) = x_1, \quad f_2(x_1,x_2) = g(x_1,x_2)h(x_1,x_2)$$

  subject to:

  $$\begin{aligned} g(x_1,x_2) &= 11 + x_2^2 - 10\cos(2\pi x_2) \\ h(x_1,x_2) &= \begin{cases} 1 - \sqrt{\frac{f_1(x_1,x_2)}{g(x_1,x_2)}} & f_1(x_1,x_2) \leq g(x_1,x_2) \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

  $$0.0 \leq x_1 \leq 1.0, \quad -30.0 \leq x_2 \leq 30.0$$

- Test Function Kursawe [21]:

  Minimize

  $$f_1(x) = \sum_{i=1}^{2}(-10\exp(-0.2 * \sqrt{x_i^2 + x_{i+1}^2})), \quad f_2(x) = \sum_{i=1}^{3}(|x_i|^{0.8} + 5\sin(x_i^3))$$

  subject to:

  $$-5.0 \leq x_1, x_2, x_3 \leq 5.0$$

- Test Function Kita [21]:

  Minimize

  $$f_1(x_1,x_2) = -x_1^2 + x_2, \quad f_2(x_1,x_2) = \frac{1}{2}x_1 + x_2 + 1$$

subject to:

$$\begin{aligned} g_1(x_1,x_2) &= \frac{1}{6}x_1 + x_2 - \frac{13}{2} \leq 0 \\ g_2(x_1,x_2) &= \frac{1}{2}x_1 + x_2 - \frac{15}{2} \leq 0 \\ g_3(x_1,x_2) &= 5x_1 + x_2 - 30 \leq 0 \end{aligned}$$

$$0.0 \leq x_1, x_2 \leq 7.0$$

- Test Function Tanaka [21]:

  Minimize

  $$f_1(x_1,x_2) = x_1, \quad f_2(x_1,x_2) = x_2$$

  subject to:

  $$\begin{aligned} g_1(x_1,x_2) &= -x_1^2 - x_2^2 + 1 + 0.1\cos(16\arctan\frac{x_1}{x_2}) \leq 0 \\ g_2(x_1,x_2) &= (x_1 - \frac{1}{2})^2 + (x_2 - \frac{1}{2})^2 - \frac{1}{2} \leq 0 \end{aligned}$$

  $$0.0 \leq x_1, x_2 \leq \pi$$

- Test Function ZDT1 [116]:

  Minimize

  $$f_1(\mathbf{x}) = x_1, \quad f_2(\mathbf{x}) = g(\mathbf{x})h(f_1,g)$$

  subject to:

  $$\begin{aligned} g(\mathbf{x}) &= 1 + 9\sum_{i=2}^{m} x_i/(m-1) \\ h(f1,g) &= 1 - \sqrt{f_1/g} \end{aligned}$$

  where $m = 30$, and $x_i \in$[0,1].

- Test Function ZDT2 [116]:

  Minimize

  $$f_1(\mathbf{x}) = x_1, \quad f_2(\mathbf{x}) = g(\mathbf{x})h(f_1,g)$$

subject to:

$$\begin{aligned} g(\mathbf{x}) &= 1+9\sum_{i=2}^{m} x_i/(m-1) \\ h(f1,g) &= 1-(f_1/g)^2 \end{aligned}$$

where $m = 30$, and $x_i \in$[0,1].

- Test Function ZDT3 [116]:

  Minimize

  $$f_1(\mathbf{x}) = x_1, \quad f_2(\mathbf{x}) = g(\mathbf{x})h(f_1,g)$$

  subject to:

  $$\begin{aligned} g(\mathbf{x}) &= 1+9\sum_{i=2}^{m} x_i/(m-1) \\ h(f1,g) &= 1-\sqrt{f_1/g}-(f_1/g)\sin(10\pi f_1) \end{aligned}$$

  where $m = 30$, and $x_i \in$[0,1].

- Test Function ZDT4 [116]:

  Minimize

  $$f_1(\mathbf{x}) = x_1, \quad f_2(\mathbf{x}) = g(\mathbf{x})h(f_1,g)$$

  subjecto to:

  $$\begin{aligned} g(\mathbf{x}) &= 1+10(m-1)+\sum_{i=2}^{m}(x_i^2-10\cos(4\pi x_i)) \\ h(f1,g) &= 1-\sqrt{f_1/g} \end{aligned}$$

  where $m = 10$, $x_1 \in$[0,1] and $x_i \in$[-5,5], $i = 2,...,m$.

- Test Function ZDT6 [116]:

  Minimize

  $$f_1(\mathbf{x}) = 1-\exp(-4x_1)\sin^6(6\Pi x_1), \quad f_2(\mathbf{x}) = g(\mathbf{x})h(f_1,g)$$

subject to:

$$\begin{aligned} g(\mathbf{x}) &= 1+9[(\sum_{i=2}^{m} x_i)/(m-1)]^{0.25}, \\ h(f_1,g) &= 1-(f_1/g)^2 \end{aligned}$$

where $m = 10$ and $x_i \in$[0,1].

- Test Function DTLZ2 [28]:

  Minimize

$$\begin{aligned} f_1(\mathbf{x}) &= [1+g(\mathbf{x})]\cos(x_1\pi/2)\cos(x_2\pi/2) \\ f_2(\mathbf{x}) &= [1+g(\mathbf{x})]\cos(x_1\pi/2)\sin(x_2\pi/2) \\ f_3(\mathbf{x}) &= [1+g(\mathbf{x})]\sin(x_1\pi/2) \end{aligned}$$

  subject to:

$$g(\mathbf{x}) = \sum_{i=3}^{m}(x_i-0.5)^2$$

  where $m = 12$ and $x_i \in$[0,1].

- Test Function DTLZ4 [28]:

  Minimize

$$\begin{aligned} f_1(\mathbf{x}) &= [1+g(\mathbf{x})]\cos(x_1^{\alpha}\pi/2)\cos(x_2^{\alpha}\pi/2) \\ f_2(\mathbf{x}) &= [1+g(\mathbf{x})]\cos(x_1^{\alpha}\pi/2)\sin(x_2^{\alpha}\pi/2) \\ f_3(\mathbf{x}) &= [1+g(\mathbf{x})]\sin(x_1^{\alpha}\pi/2) \end{aligned}$$

  subject to:

$$g(\mathbf{x}) = \sum_{i=3}^{m}(x_i-0.5)^2$$

  where α=100, $m = 12$ and $x_i \in$[0,1].

- Test Function DTLZ6 [28]:

  Minimize

$$\begin{aligned} f_1(\mathbf{x}) &= x_1 \\ f_2(\mathbf{x}) &= x_2 \\ f_3(\mathbf{x}) &= [1+g(\mathbf{x})]h(f_1,f_2,g) \end{aligned}$$

subject to:

$$g(\mathbf{x}) = 1+9/(m-2)\sum_{i=3}^{m} x_i,$$

$$h(f_1, f_2, g) = 3-\sum_{i=1}^{2}[\frac{f_i}{1+g}(1+\sin(3\pi f_i))]$$

where $m = 22$ and $x_i \in [0,1]$.

2.4.4 Measures of Performance

In this section, we provide the definitions of the measures of performance that are going to be used to validate the different approaches proposed as part of this work.

- **Error Ratio (ER)**: This measure was proposed by Van Veldhuizen [105] to indicate the percentage of solutions (from the nondominated vectors found so far) that are not members of the true Pareto optimal set:

$$\text{ER} = \frac{\sum_{i=1}^{n} e_i}{n}$$

where n is the number of vectors in the current set of nondominated vectors available; $e_i = 0$ if vector i is a member of the Pareto optimal set, and $e_i = 1$ otherwise. It should then be clear that ER$= 0$ indicates an ideal behavior, since it would mean that all the vectors generated by our algorithm belong to the true Pareto optimal set of the problem.

- **Success Counting (SCC)**: We define this measure based on the idea of the measure called *Error Ratio* proposed by Van Veldhuizen [105] which indicates the percentage of solutions (from the nondominated vectors found so far) that are not members of the true Pareto optimal set. In this case, we count the number of vectors (in the current set of nondominated vectors available) that are members of the Pareto optimal set:

$$\text{SCC} = \sum_{i=1}^{n} s_i$$

where n is the number of vectors in the current set of nondominated vectors available; $s_i = 1$ if vector i is a member of the Pareto optimal set, and $s_i = 0$

otherwise. It should then be clear that SCC= n indicates an ideal behavior, since it would mean that all the vectors generated by our algorithm belong to the true Pareto optimal set of the problem. For a fair comparison, when we use this measure, all the algorithms should limit their final number of nondominated solutions to the same value.

- **Generational Distance (GD)**: The concept of generational distance was introduced by Van Veldhuizen & Lamont [106] as a way of estimating how far are the elements in the Pareto front produced by our algorithm from those in the true Pareto front of the problem. This measure is defined as:

$$\text{GD} = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n}$$

  where n is the number of nondominated vectors found by the algorithm being analyzed and d_i is the Euclidean distance (measured in objective space) between each of these and the nearest member of the true Pareto front. It should be clear that a value of GD= 0 indicates that all the elements generated are in the true Pareto front of the problem. Therefore, any other value will indicate how "far" we are from the global Pareto front of our problem.

- **Inverted Generational Distance (IGD)**: As described previously, the GD measure estimates how far are the elements in the Pareto front produced by our algorithm from those in the true Pareto front of the problem. In our case, we implemented an "Inverted" Generational Distance measure (IGD) in which we use as a reference the true Pareto front, and we compare each of its elements with respect to the front produced by an algorithm. In this way, we are calculating how far are the elements of the true Pareto front, from those in the Pareto front produced by our algorithm. Computing this IGD value reduces the bias that can arise when an algorithm didn't fully cover the true Pareto front. It should be clear that a value of IGD= 0 indicates that the Pareto front obtained by our algorithm covers the whole true Pareto front. Therefore, any other value will indicate how "far" is the true Pareto front from the Pareto front obtained by our approach.

- **Spacing (SP)**: This measure was proposed by Schott [94] as a way of measuring the range (distance) variance of neighboring vectors in the Pareto

front known. This measure is defined as:

$$\mathrm{SP} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\bar{d} - d_i)^2}$$

where $\underline{d}_i = min_j(\sum_{k=1}^{m} |f_m^i - f_m^j|)$, $i, j = 1, ..., n$, m is the number of objectives, $\bar{d}$ is the mean of all d_i, and n is the number of vectors in the Pareto front found by the algorithm being evaluated. A value of zero for this measure indicates that all the nondominated solutions found are equidistantly spaced.

- **Two Set Coverage (SC)**: This measure was proposed in [116], and it can be termed *relative coverage comparison of two sets*. Consider X', X'' as two sets of vectors. SC is defined as the mapping of the order pair (X', X'') to the interval $[0, 1]$:

$$\mathrm{SC}(X', X'') = |\{a'' \in X''; \exists a' \in X' : a' \prec a''\}|/|X''|$$

If all points in X' dominate or are equal to all points in X'', then by definition $\mathrm{SC}(X', X'') = 1$. $\mathrm{SC}(X', X'') = 0$ implies the opposite. In general, $\mathrm{SC}(X', X'')$ and $\mathrm{SC}(X'', X')$ both have to be considered due to set intersections not being empty. When $\mathrm{SC}(X', X'') = 1$ and $\mathrm{SC}(X'', X') = 0$, we say that X' is better than X''.

- **Two Set Difference Hypervolume (HV)**: This measure was proposed in [115]. Consider X', X'' as two sets of vectors. HV is defined by:

$$\mathrm{HV}(X', X'') = \delta(X' + X'') - \delta(X'')$$

where the set $X' + X''$ is defined as the nondominated vectors obtained from the union of X' and X'', and δ is the unary hypervolume measure. $\delta(X)$ is defined as the hypervolume of the portion of the objective space that is dominated by X. In this way, $\mathrm{HV}(X', X'')$ gives the hypervolume of the portion of the objective space that is dominated by X' but not for X''. We use the origin as a reference point to compute the hypervolume. So, since all the test functions used have to be minimized, with this measure we obtain a difference between the areas that *dominate* the analyzed Pareto fronts. In this way, if $\mathrm{HV}(X', X'') = 0$ and $\mathrm{HV}(X'', X') < 0$, we say that X'' is better than X'.

Chapter 3

Coevolutionary Multi-Objective Optimization

As the first part of our work, we proposed a multi-objective evolutionary algorithm that incorporates coevolutionary concepts. Such approach is described in this chapter.

3.1 Coevolution

We call coevolution to a change in the genetic composition of a species (or group of species) as a response to a genetic change of another one. In a more general sense, coevolution refers to a reciprocal evolutionary change between species that interact with each other. The term "coevolution" is usually attributed to Ehrlich and Raven who published a paper on their studies performed with butterflies and plants in the mid-1960s [30].

The relationships between the populations of two different species A and B can be described considering all their possible types of interactions. Such interaction can be positive or negative depending on the consequences that such interaction produces on the population. All the possible interactions between two different species are shown in Table 3.1.

Evolutionary computation researchers have developed several coevolutionary approaches in which normally two or more species relate to each other using one of the previously indicated schemes. Also, in most cases, such species evolve independently through a genetic algorithm. The key issue in these coevolutionary algorithms is that the fitness of an individual in a population depends on the

	A	B	
Neutralism	0	0	Populations A and B are independent and don't interact.
Mutualism	+	+	Both species benefi t from the relationship.
Commensalism	+	0	One species benefi ts from the relationship but the other is neither harmed nor benefi ted.
Competition	-	-	Both species have a negative effect on each other since they are competing for the same resources.
Predation	+	-	The predator (A) benefi ts while the prey (B) is negatively affected.
Parasitism	+	-	The parasite (A) benefi ts while the host (B) is negatively affected.

Table 3.1: All the possible interactions between two different species.

individuals of a different population. In fact, we can say that an algorithm is coevolutionary if it has such property. There are two main classes of coevolutionary algorithms in the evolutionary computation literature:

- Those based on *competition* relationships: In this case, the fitness of an individual is the result of a series of "encounters" with other individuals [73, 85].
- Those based on *cooperation* relationships: In this case, the fitness of an individual is the result of a collaboration with individuals of other species (or populations) [76, 75].

3.2 Related Work

There are very few references in the literature in which coevolutionary concepts are used to solve multi-objective optimization problems. We will review the main ones in this section.

- Parmee & Watson [74] proposed a collaborative scheme in which they use one population to optimize each of the objective functions of a problem. The authors propose to use individual GAs for the optimization of each objective. For each generation, solutions relating to each objective are compared with the best individual from the other GA populations. If a variable is outside a defined range, it is adjusted by a penalty function. The method is

really designed to converge to a single (ideal) trade-off solution. However, through the use of penalties the algorithm can maintain diversity in the population. These penalties relate to variability in the decision variables values. The authors also store solutions produced during the evolutionary process so that the user can analyze the historical paths traversed by the algorithm.

- Jiangming Mao et. al. [62] applied the symbiotic genetic algorithm (SGA) [43] to multi-objective optimization problems. This approach uses a single population in which individuals are ranked based on Pareto dominance. Then their fitnesses are modified based on two factors: (1) the symbiotic factor θ_{ij} (which represents the symbiotic relationships between individuals i and j), and (2) the factor θ_{ml} (which represents the symbiotic relationships between the objective functions m and l). The SGA is structured in such a way that, besides the GA loop, there is an outer loop for training symbiotic relations, which includes the calculation of symbiotic parameters, through fuzzy inference rules. Training of parameters in the fuzzy inference leads to obtaining the required distribution of individuals. With this aim, authors incorporate user's preferences through an aggregating function, used as objective function in the training process.

- Barbosa and Barreto [5] proposed a cooperative approach for solving a graph layout generation problem. The approach uses two populations (a separate genetic algorithm is used for each of them and information is exchanged through a shared fitness function): a graph layout population (i.e., individuals that contain the coordinates of all vertices in the graph) and a population of weights (i.e., individuals that contain, each one, a set of weights to be applied on the different aesthetic objectives imposed on the problem). Each of the solutions produced by the system are presented to a user who ranks them based on (subjective) preferences. This ranking is used to determine fitness of the population of weights.

- Keerativuttitumrong et. al. [54], Tan et.al. [101] and Iorio and Li [49], proposed cooperative schemes in which one population is defined for each decision variable of the problem. In order to evaluate an individual in any population, individuals from the other populations must be selected in order to complete a solution. In [54], the evolution of each of these populations is controlled through Fonseca and Fleming's MOGA [36]. The method in [101] uses an external archive to store and update the nondominated solutions found so far and also to guide the search to the less exploited sub-

regions of the search space. Finally, in [49] the evolution of each of the populations is controlled through the scheme of NSGA-II [27]. After each generation, the method uses a nondominated sorting over all the subpopulations of parents and offspring to determine the new parents subpopulations.

As we could see, some of the approaches previously described need some previous knowledge about the problem (like [74]) or a large set of parameters (like [62]). Also, there are approaches, like [5], that depend on an interactive feedback of the user to fix the value of the parameters needed. Finally, it is very important to note that the main feature of the cooperative approaches (like [54, 101, 49]) is that they optimize the different variables (or sets of them) of the problem by using different populations. For this reason, the main disadvantage of these approaches is that the possible interdependence among variables (also called *epistasis*) may cause some difficulties when the variables are being optimized in a separate way.

In the next section, we describe a coevolutionary approach that doesn't need any kind of previous knolewdge of the problem or feedback of the user. Additionally, although the proposed approach also subdivides the search space (similar to a cooperative scheme), it does such work by using the information obtained throughout the evolutionary process.

3.3 Coevolutionary Multi-Objective Approach

We proposed a Coevolutionary Multi-Objective Evolutionary Algorithm, called CO-MOEA, whose main idea is to obtain information along the evolutionary process in order to focus the search efforts only on the promising subregions of the search space, while ignoring the useless subregions and reducing the computational cost involved in the process of convergence to the optimal solutions.

Our approach subdivides the search space into n subregions, and then it uses a subpopulation for each of these subregions. At each generation, these different subpopulations "cooperate" and "compete" among themselves and from these different processes we obtain a single Pareto front. The size of each subpopulation is adjusted based on their contribution to the current Pareto front. In order to determine what regions of the search space are promising, our algorithm performs an analysis of the current Pareto front.

In this section, we will describe the three versions of our approach, that represent the three stages of the process followed to obtain the final version of our algorithm. As we mentioned before, our approach incorporates the use of several populations, obtained by means of a mechanism used to subdivide the search

space. Thus, one of the main features of our algorithm must be the scalability, since the number of possible different populations may grow exponencially with the number of variables of the problem that we want to solve. In this way, as we will see, the second and third versions of our algorithm incorporate some improvements made with the aim to reduce the scalability problems of the final version of our approach.

3.3.1 First Version

The evolutionary process of the first version of our approach was divided into 4 stages. The change of stage was controlled by a certain number of generations during which we run the algorithm. The full evolutionary run was divided into four parts (i.e., the total number of generations is divided by four), and each stage was allocated on one of these four parts. Figure 3.1 shows the pseudocode of the first version of our algorithm.

First Stage. During the first stage (first 25% of the total number of generations), the algorithm is allowed to explore all of the search space, by using a population of individuals that evolves using Fonseca and Fleming's MOGA [36]. Additionally, the approach uses the adaptive grid proposed by [57]. At the end of this first stage, the algorithm analyses the current Pareto front (stored in the adaptive grid) in order to determine what variables of the problem are more critical. In this first version, this analysis consisted of looking at the current values of the decision variables corresponding to the current Pareto front (line 6, Figure 3.1). This analysis was performed independently for each decision variable. The idea was to determine if the values corresponding to a certain variable were distributed along all the allowable interval or if such values were concentrated on a narrower range. When the whole interval was being used, the algorithm kept the entire interval for that variable. However, if only a narrow portion was being used, then the algorithm tried to identify portions of the interval that could be discarded from the search process. As a result of this analysis, the algorithm determined whether was convenient or not to subdivide (and, in such case, it also determined how many subdivisions should be created) the interval of a certain decision variable. Each of these different regions were assigned a different population (line 7, Figure 3.1).

We illustrate this process with an example. Let us suppose that our problem has two variables and that, after the analysis, the algorithm determines that it is not convenient to subdivide the interval of the first variable. Additionally, the algorithm determines that the interval of the second variable must have two subdivisions. What the algorithm does is to divide the interval of the second decision

```
    Begin
        Initialize population
1.      gen ← 1
2.      populations ← 1
3.      While (gen ≤ Gmax)
4.          If(!gen%(Gmax/4))
5.              check_active_populations()
6.              decision_variables_analysis()
                (compute number of subdivisions)
7.              construct_new_subpopulations()
                (update populations)
            EndIf
8.          For (i ← 1; i ≤ populations; i ← i + 1)
9.              If (population i contributes
                to the current Pareto front)
10.                 evolve_and_compete(i)
                EndIf
            EndFor
11.         elitism()
12.         reassign_resources()
13.         gen ← gen + 1
        EndWhile
        Report results in current Pareto front
    End
```

Figure 3.1: Pseudocode of the first version of our algorithm.

variable into three parts of equal size (i.e., add two subdivisions to the interval). The process to decide how many populations to have and to which region of the search space to assign each of them is illustrated in Figure 3.2.

Second Stage. When reaching the second stage, the algorithm consists of a certain number of populations looking each at different regions of the search space. At each generation, the evolution of all the populations takes place independently and, later on, the nondominated elements from each population are stored in the adaptive grid where they "cooperate" and "compete" in order to conform a single Pareto front (line 10, Figure 3.1). After this, we count the number of individuals that each of the populations contributed to the current Pareto front. Our algorithm is *elitist* (line 11, Figure 3.1), because after the first generation of the second stage, all the populations that do not provide any individual to the current Pareto front are automatically eliminated and the sizes of the other populations

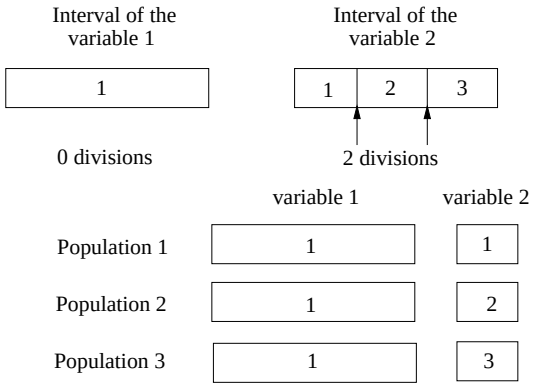


Figure 3.2: Mechanism used to assign regions of the search space to each population.

are properly adjusted (i.e., those populations that contributed more to the current Pareto front get their sizes proportionally increased and those who contribute less get their sizes decreased; line 12, Figure 3.1). Thus, populations contributing with more individuals to the current Pareto front are "rewarded" with a number of extra individuals which is proportional to the percentage contributed to the current Pareto front. Individuals to be added or removed are randomly generated/chosen. Thus, populations compete with each other to get as many extra individuals as possible. This process is illustrated in Figure 3.3. Figure 3.4 illustrates the second stage of our algorithm.

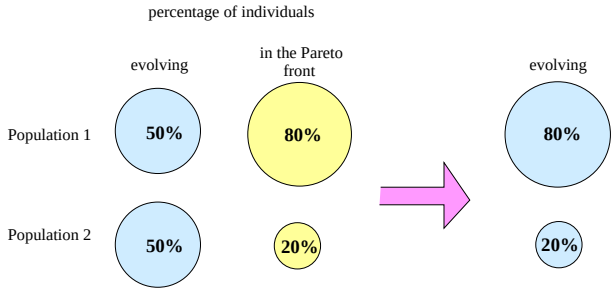


Figure 3.3: Resources reassignment: Each population is assigned or removed individuals such that its final size is proportional to its contribution to the current Pareto front.

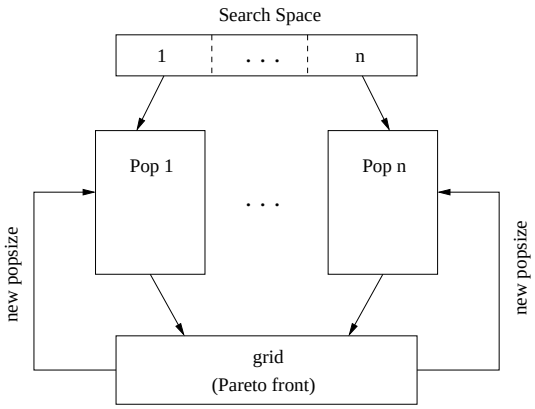


Figure 3.4: Graphical representation of the second stage of our algorithm.

Third Stage. During the third stage, we perform a check on the current populations in order to determine how many (and which) of them can continue (i.e., those populations which continue contributing individuals to the current Pareto front) (line 5, Figure 3.1). Over these (presumably good) populations, we will apply the same process from the second stage (i.e., they will be further subdivided and more populations will be created in order to exploit these "promising regions" of the search space).

In order to determine the number of subdivisions that are to be used during the third stage, we repeat the same analysis as before (i.e., the analysis performed during the first stage). The individuals from the "good" populations are kept. All the good individuals are distributed across the newly generated populations. A new count is undertaken so that the algorithm can determine how many individuals are contributed by each of the new populations to the current Pareto front. Again, populations that do not contribute to the current Pareto front are eliminated. Note however, that we define a minimum population size and this size is enforced for all populations at the beginning of the third stage. After the first generation of the third stage, the size will be adjusted based on the same criteria as before (i.e., the size of populations will be modified based on their contribution to the current Pareto front).

Fourth Stage. During this stage, we apply the same procedure of the third stage in order to allow a fine-grained search.

Decision Variables Analysis. The mechanism adopted for the decision variables analysis was very simple. Given a set of values within an interval, we

computed both the minimum average distance of each element with respect to its closest neighbor and the total portion of the interval that was covered by the individuals contained in the current Pareto front. Then, only if the set of values covered less than 80% of the total of the interval, the algorithm proceeded to divide it. In this case, the number of divisions got increased (without exceeding a total of 10 divisions per interval), as explained next. Let us define *range* as the percentage of the total of $\underline{interval}$ that is occupied by the values of the variable under consideration. Let $\bar{d}_{min}$ be the minimum average distance between individuals and let *divisions* be the number of divisions to perform in the interval of the variable:

If $(range < 0.\underline{8} * interval)$
  While $(\bar{d}_{min} < 0.2 * interval)$
    $divisions \leftarrow divisions + 1;$
    $interval \leftarrow 0.2 * interval;$
  EndWhile
EndIf

The results of this approach can be consulted in [16]. In general, the results of our algorithm in low-dimensional functions were of good quality. However, when we tested our approach using functions with search spaces with higher dimension, we started to have problems with the number of populations that it could potentially need to handle. That is, the first version of our algorithm had scalability problems. In this way, we considered to redesign the algorithm so that such multiple populations were no longer needed. Also, we considered the use of a clustering algorithm to determine the most critical decision variables of the problem (replacing the current analysis used). These improvements gave place to the second version of our algorithm.

3.3.2 Second Version

The second version of our algorithm performs a clustering analysis on the set of decision variables of the current Pareto front in order to identify promising regions of the search space. In this way, the number of populations needed does not grow exponentially with the number of decision variables as in our original proposal. In fact, the number of populations needed does not exceed the total number of members on the true Pareto front.

Figure 3.5 shows the pseudocode of the second version of our algorithm. In this version, the evolutionary process of our approach is only divided into two

main stages. The first stage works in a similar way that the previous version, but in this case we replace the decision variables analysis described before with a clustering analysis, in order to determine the promising regions of the search space.

```
    Begin
        Initialize population
1.      gen ← 1
2.      populations ← 1
3.      While (gen ≤ Gmax)
            If (gen ≥ Gmax/4)
4.                  If (!gen%(Gmax/4) or
                    ∃ x ∈ pop_zero : x ∈ current Pareto front)
5.                          check_active_populations()
6.                          clustering_algorithm()
7.                          construct_new_subpopulations()
                    EndIf
            EndIf
8.          For (i ← 1; i ≤ populations; i ← i + 1)
9.                  If (population i contributes to the current Pareto front)
10.                         evolve_and_compete(i)
                    EndIf
            EndFor
11.         elitism()
12.         reassign_resources()
13.         gen ← gen + 1
        EndWhile
        Report results in current Pareto Front
    End
```

Figure 3.5: Pseudocode of the second version of our algorithm.

Thus, we perform a clustering analysis on the set of values of the decision variables corresponding to the current Pareto front. This analysis is performed independently for each decision variable. Once we know the clusters corresponding to each one of the decision variables, we proceed to form a set of new populations. Each cluster provides a specific interval. Then, a set of sub-regions is created as a cartesian product of the set of the intervals corresponding to the clusters of each variable. After that, we assign a new population to those sub-regions that have individuals in the current Pareto front (line 7, Figure 3.5). This process is illustrated in Figure 3.6.

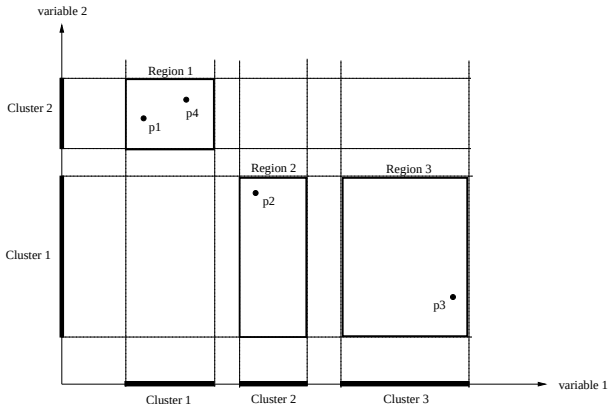


Figure 3.6: Mechanism used to locate the promising regions of the search space. A population will be assigned to each located promising region.

Finally, we use one extra population (called population zero) that is searching for good solutions on the sub-regions that are not being explored by the other populations (for this purpose we use a special mutation operator that keeps the individuals from belonging to the sub-regions explored by the other populations).

The second stage of the algorithm consists of the second, third and fourth stages of the previous version. That is, we join those three stages in just one. In this version, we have called *checkpoint* to the process made in the previous version at the beginning of the third and fourth stages:

Checkpoint. During the second stage, we perform a *checkpoint* at specific moments of the evolutionary process (line 4, Figure 3.5). The checkpoint takes place as before (see Figure 3.1), but also when the population zero includes a new individual in the current Pareto front. When the checkpoint occurs, we perform a check on the current populations in order to determine how many (and which) of them can continue (line 5, Figure 3.5). As at the end of the first stage, we perform again the clustering analysis on the set of values of the decision variables corresponding to the current Pareto front, and proceed to form a set of new populations.

Clustering Analysis. We implemented a clustering algorithm based on the nature of the $k-means$ algorithm [50]. This algorithm begins with k random centroids and puts every point of the analyzed set on the cluster corresponding to the nearest centroid. After that, the *means* of each cluster are calculated and considered as the new centroids, and the process is repeated until no further changes are done. The algorithm stops when the minimum of the sum of the distances between each point

to its corresponding centroid is found. This algorithm has two disadvantages: (1) it depends on the initial centroids and (2) it requires the number of clusters needed. For this reason, we made two modifications to overcome these drawbacks. Regarding the first disadvantage, we look for a point that could be a new centroid: Let x_i a point that belongs to a cluster with centroid c_i, and d_{min} the minimum distance between two centroids. If $d(x_i, c_i) > d_{min}$, x_i will be a new centroid. To maintain the number of clusters constant, once we have selected a point to be a new centroid, we choose one of the closest centroids to be eliminated. With respect to the second disadvantage, we use the following mechanism [50]: Let x_i be a point that belongs to cluster q (with centroid c_q) and K the current total number of clusters. The average distance between x_i and the K centroids is: $\bar{d_i} = (1/K)\sum_{k=1}^{K} d(x_i, c_k)$. We create a new cluster with centroid x_i when: $|d(x_i, c_q) - \bar{d_i}| \leq \bar{d_i}T$ where T is such that $0 < T < 1$. The number of clusters created is proportional to the value of T. Since the previous mechanism creates new clusters, we also eliminate clusters when the corresponding centroids are very close: If the distance between two centroids is less that T times the average distance between centroids, one of them is eliminated.

It is worth emphasizing that the clustering technique used was taken from the specialized literature for being one of the most popular approaches in current use and representative of the state-of-the-art. However, the use of a different (perhaps better, under certain conditions) technique is always possible for this purpose, as for example the subtractive clustering technique proposed by Chiu in 1994 [13].

The results of this approach can be consulted on [83]. Although the results obtained by the second version of our approach were better than the obtained by the previous one, in this case we still had scalability problems. However, in this case the problem was not the number of populations actually needed, but the number of populations virtually constructed. Thus, the main drawback of this version was the procedure followed for constructing new populations. As we described before, in such process we constructed virtually all the possible regions given by the cartesian product of the set of intervals corresponding to the clusters of each variable. When we tested the algorithm with functions with high dimensional decision spaces, once again the problem was the number of virtual regions that it needed to handle. For this reason, we decided to improve the construction procedure, and this update gave place to the third and final version of our approach.

Parameters Required. Our proposed approach requires the following parameters (the parameter T of the clustering algorithm used was fixed to $T = 1$):

1. Crossover rate (p_c) and mutation rate (p_m).

2. Maximum number of generations (*Gmax*).

3. Size of the initial population ($popsize_{init}$) to be used during the first stage and minimum size of the secondary population ($popsize_{sec}$) to be used during the further stages.

3.3.3 Third Version

In this version, the set of sub-regions is created in the following way. Once we know the intervals corresponding to the clusters obtained for each variable, for each point in the current Pareto front, we proceed to locate the interval on each variable to which it belongs. This process gives us a region in the search space. Then, for each point in the current Pareto front, we first check if it belongs to any region already located. If the point belongs to an existing region, we continue with the next point. Otherwise, we proceed to create the corresponding region. And so on. After that, we assign a new population to each region created, i.e., those that have individuals in the current Pareto front (line 7, Figure 3.5). In this way, in the worst case we will have as many populations as points in the current Pareto front.

On the other hand, in this version we decided to initialize the population *zero* (described before) with an 80% of points of the current Pareto front and a 20% of random points (with the aim of generating intermediate points on the current Pareto front while adding diversity).

Finally, for the final version of our approach, we also decided to change the representation used. The binary representation used before was replaced by a real numbers representation. This change improved the obtained results.

3.3.4 Coevolutionary Interactions

In this section, we would like to discuss the coevolutionary interactions implicit in our three proposed approaches. As we mentioned before, at each generation of the second stage, the populations (considered as separate species[1]) "cooperate" and "compete" (through their nondominated elements) in order to conform a single Pareto front. As a result of this competition/collaboration process, the size of the populations is adjusted in such a way that the populations assigned to the promising regions of the search space become larger, while the populations assigned to

[1] In this case, we consider each population as a different species.

the worst regions of the search space may even extinguish (be eliminated). In this way, every population competes to provide the largest possible number of individuals to the final Pareto front, in order to be able to increase its size and, as a consequence, its survival possibilities. These coevolutionary interactions are indicated in the diagram shown in Figure 3.7.

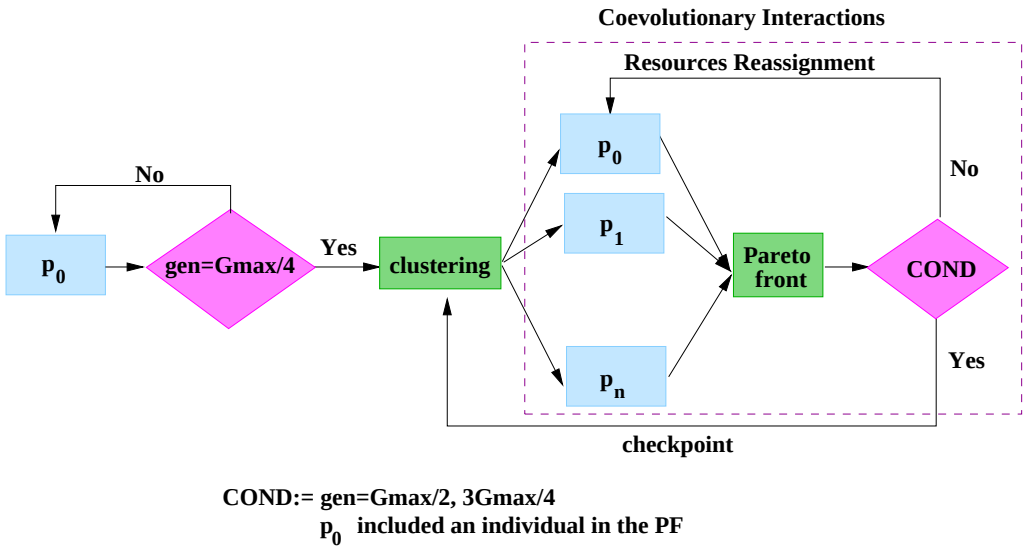


Figure 3.7: Diagram of the final version of our algorithm.

On the other hand, although this is in fact a competition process between individuals (where those nondominated win), the fitness of each individual is affected by the competition results of its whole species. This is due to the use of *fitness sharing*, incorporated in the MOGA approach (used as a search engine by our coevolutionary scheme). As we mentioned in Section 2.4.1, when fitness sharing is used, the fitness of an individual is degraded in proportion to the number and closeness to individuals that belong to its same niche. See Figure 3.8.

Thus, we can see that the main coevolutionary interaction incorporated into our approach is the competition between species in order to conform a single Pareto front, since the fitness of each individual within each one of the populations depends of the resources reassignment determined by the results of the competition process.

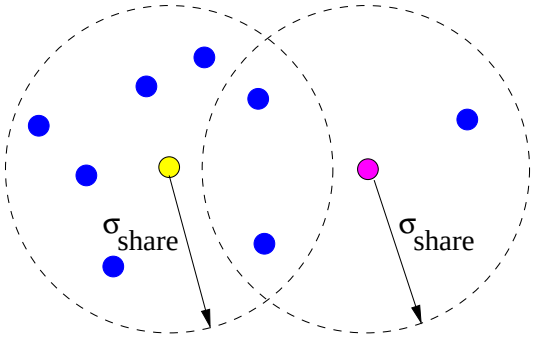


Figure 3.8: For each individual, a niche is defined. The fitness of an individual is degraded in proportion to the number and closeness to individuals that belong to its same niche.

3.4 Results

We present the results obtained by the third and final version of our approach [96].

We used the methodology normally adopted in the evolutionary multi-objective optimization literature [21]. We performed both quantitative comparisons (adopting four measures of performance) and qualitative comparisons (plotting the Pareto fronts produced) with respect to two multi-objective evolutionary algorithms (MOEAs) that are representative of the state-of-the-art in the area: the Nondominated Sorting Genetic Algorithm II (NSGA-II) [27], and the Strength Pareto Evolutionary Algorithm 2 (SPEA2) [118].

For our comparative study, we implemented four measures of performance: Error Ratio (ER), Generational Distance (GD), Spacing (SP) and Two Set Coverage (SC). All these measures are defined in Section 2.4.4, page 25. Also, we used four different test functions: Deb, Kursawe, Kita and Tanaka (defined in Section 2.4.3, page 19).

For each of the test functions, we performed 30 runs per algorithm and a total of 10,000 evaluations. The parameters for NSGA-II were *popsize*=100 and 100 generations and for SPEA2 were $\alpha = \mu = \lambda = 100$ and 100 generations. All the algorithms used real numbers representation, a bit mutation probability (p_m) equal to $1/codesize$ (Parameter Based Mutation) and a crossover probability (p_c) equal to 0.8 (Simulated Binary Crossover). The Pareto fronts that we will show correspond to the median of the 30 runs with respect to the ER measure.

Regarding constraint-handling, we used the original scheme provided in the

case of the NSGA-II. However, since the other two algorithms (including our own) didn't have a constraint-handling mechanism, we implemented for them a simple dynamic penalty function (based on the number of generations) over the value of the objective functions of each infeasible individual.

Test Function Deb

In this example, our approach used: $popsize_{init} = 100$, $popsize_{rec} = 30$ (38 gen). Table 3.2 shows the values of the measures for each of the MOEAs compared.

Test Function Deb				
		CO-MOEA	NSGA-II	SPEA2
ER	best	0.02	0.00	0.02
	median	0.10	0.07	0.05
	worst	0.44	0.47	0.39
	average	0.15	0.13	**0.08**
	std. dev.	0.1112	0.1289	0.0856
GD	best	0.0001	0.0047	0.0048
	median	0.0040	0.0056	0.0056
	worst	0.0910	0.0061	0.6116
	average	0.0159	**0.0055**	0.0829
	std. dev.	0.0249	0.0004	0.1303
SP	best	0.0045	0.0064	0.0027
	median	0.0090	0.0073	0.0041
	worst	0.9069	0.0084	1.9915
	average	0.1344	**0.0073**	0.4252
	std. dev.	0.2491	0.0006	0.6227

Two Set Coverage Metric SC				
	X	SC(X,I)	SC(X,II)	SC(X,II)
I	CO-MOEA	0.00	0.00	0.00
II	NSGA-II	0.02	0.00	0.01
III	SPEA2	0.00	0.03	0.00
Average		1%	2%	**0%**

Table 3.2: Comparison of results between our approach (denoted by CO-MOEA), the NSGA-II [27] and the SPEA2 [118] for test function Deb.

As we can see in Table 3.2, the three algorithms have similar results with respect to the ER measure. However, the SPEA2 algorithm has the best results on

average. In the case of the GD and SP measures, although our algorithm and the SPEA2 algorithm, respectively, have better best and median results than the other algorithms, the best results on average are from the NSGA-II algorithm in both cases given its low standard deviation. Finally, regarding the SC measure all the algorithms have very similar results (between 0 and 2%). The Pareto fronts for this function are shown in Figure 3.9.

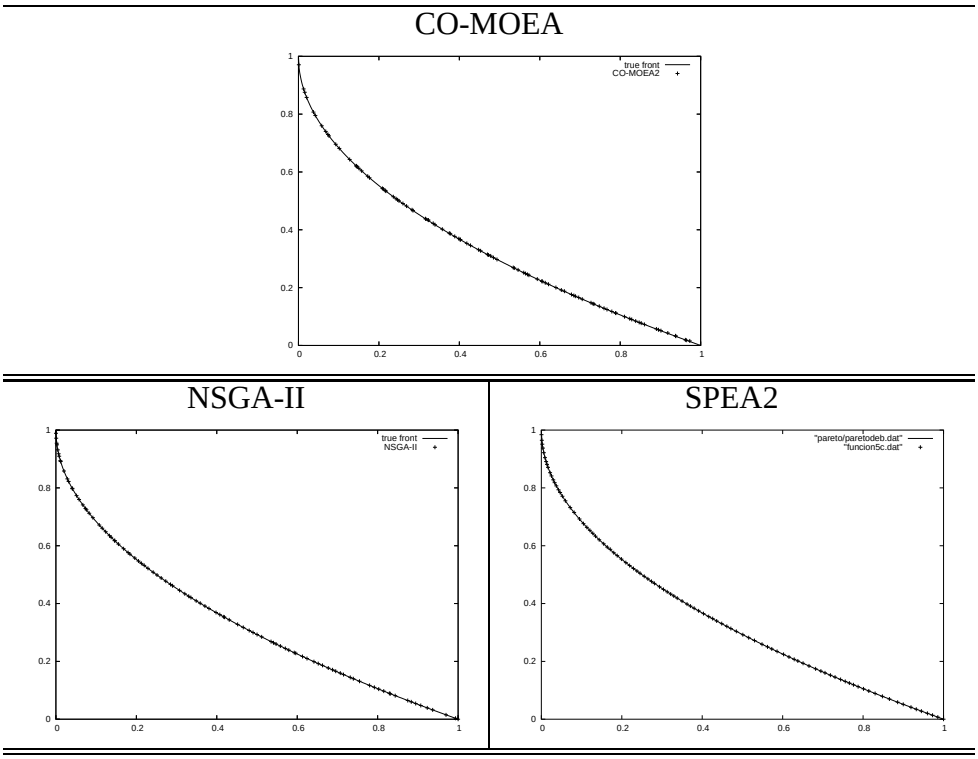


Figure 3.9: Pareto fronts obtained by our approach (CO-MOEA), the NSGA-II [27] and the SPEA2 [118], for test function Deb.

Test Function Kursawe

In this case, our approach used: $popsize_{init} = 100$, $popsize_{rec} = 30$ (40 gen). Table 3.3 shows the values of the measures for each of the MOEAs compared.

With respect to the ER and GD measures (see Table 3.3), the three algorithms have very similar results, being our algorithm marginally the best. Regarding the SP measure, the NSGA-II algorithm has the best results. Finally, in this case with

Test Function Kursawe				
		CO-MOEA	NSGA-II	SPEA2
ER	best	0.12	0.16	0.19
	median	0.23	0.27	0.26
	worst	0.35	0.37	0.36
	average	**0.24**	0.28	0.25
	std. dev.	0.0578	0.0578	0.0412
GD	best	0.0028	0.0032	0.0028
	median	0.0032	0.0036	0.0033
	worst	0.0038	0.0044	0.0035
	average	**0.0032**	0.0037	**0.0032**
	std. dev.	0.0002	0.0004	0.0002
SP	best	0.0519	0.0450	0.0991
	median	0.1100	0.0553	0.0867
	worst	0.1534	0.1060	0.0801
	average	0.1069	**0.0606**	0.0866
	std. dev.	0.0306	0.0156	0.0042

Two Set Coverage Metric SC				
	X	SC(X,I)	SC(X,II)	SC(X,II)
I	CO-MOEA	0.00	0.07	0.17
II	NSGA-II	0.07	0.00	0.14
III	SPEA2	0.07	0.09	0.00
Average		**7%**	8%	16%

Table 3.3: Comparison of results between our approach (denoted by CO-MOEA), the NSGA-II [27] and the SPEA2 [118] for test function Kursawe.

respect to the SC measure, our algorithm has the best results with an average of 7% of its points dominated by other algorithms. In this case, the percentages of NSGA-II and SPEA2 are: 8% and 16%, respectively. The Pareto fronts for this function are shown in Figure 3.10.

Test Function Kita

In this example, our approach used: $popsize_{init} = 100$, $popsize_{rec} = 30$ (40 gen). Table 3.5 shows the values of the measures for each of the MOEAs compared.

In this function, there are clear differences between the three algorithms in the values of the ER measure: the best is SPEA2 followed by NSGA-II and our algorithm (see Table 3.4). With respect to the GD measure, although our algorithm

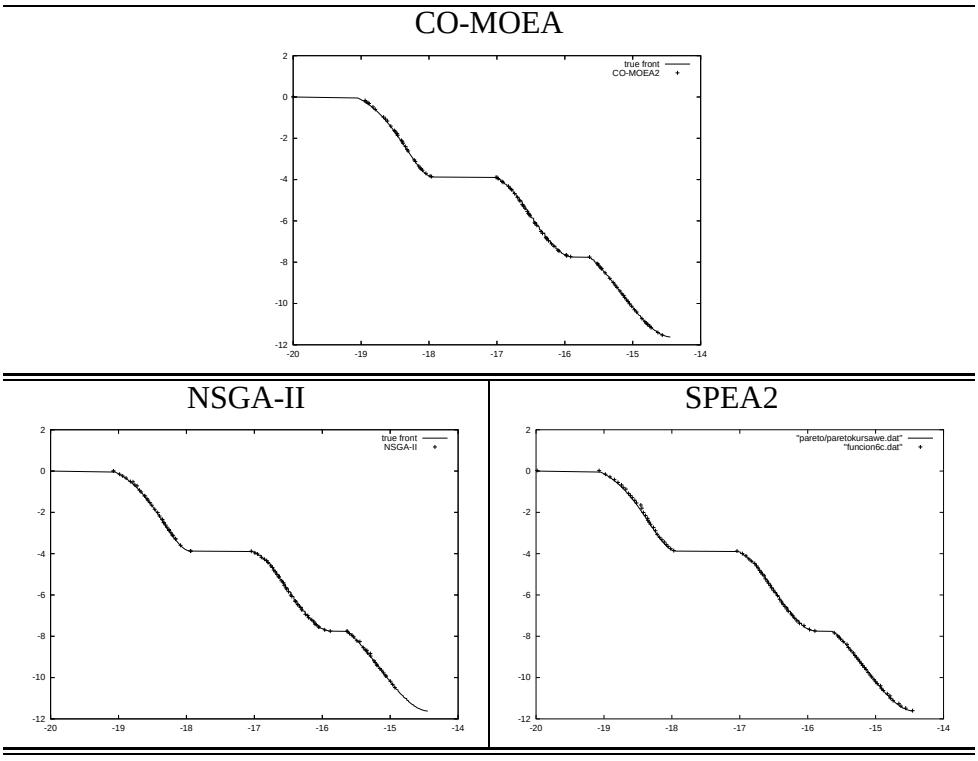


Figure 3.10: Pareto fronts obtained by our approach (CO-MOEA), the NSGA-II [27] and the SPEA2 [118], for test function Kursawe.

has the best value, on average the SPEA2 has the best results, followed by our algorithm and the NSGA-II. In the case of the SP measure, although the better best and median results are from SPEA2 and NSGA-II, the best results on average are from our algorithm given its low standard deviation. Regarding the SC measure, SPEA2 has the best results with an average of 11% of its points dominated by other algorithms. In this case, the percentages of NSGA-II and our algorithm are: 22% and 34%, respectively. The Pareto fronts for this function are shown in Figure 3.11.

Test Function Tanaka

In this example, our approach used: $popsize_{init} = 100$, $popsize_{rec} = 30$ (40 gen). Table 3.5 shows the values of the measures for each of the MOEAs compared.

As we can see in Table 3.5, the best results on average with respect to the ER

Test Function Kita				
		CO-MOEA	NSGA-II	SPEA2
ER	best	0.37	0.30	0.27
	median	0.52	0.44	0.33
	worst	0.67	0.58	0.43
	average	0.51	0.46	**0.35**
	std. dev.	0.0793	0.0624	0.0455
GD	best	0.0025	0.0019	0.0026
	median	0.0056	0.0025	0.0038
	worst	0.1980	0.5224	0.1569
	average	0.0239	0.0627	**0.0222**
	std. dev.	0.0411	0.1412	0.0395
SP	best	0.0351	0.0192	0.0190
	median	0.0539	0.0248	0.0284
	worst	0.1868	2.8625	1.3698
	average	**0.0636**	0.1513	0.1504
	std. dev.	0.0295	0.5286	0.3025

Two Set Coverage Metric SC				
	X	SC(X,I)	SC(X,II)	SC(X,II)
I	CO-MOEA	0.00	0.15	0.14
II	NSGA-II	0.28	0.00	0.07
III	SPEA2	0.39	0.29	0.00
Average		34%	22%	**11%**

Table 3.4: Comparison of results between our approach (denoted by CO-MOEA), the NSGA-II [27] and the SPEA2 [118] for test function Kita.

measure are from SPEA2, followed by NSGA-II and our algorithm. With respect to the GD measure, the results of the three algorithms are almost equal. In the case of the SP measure, the best results on average are from SPEA2, followed by our algorithm and NSGA-II. Regarding the SC measure, NSGA-II has the best results with an average of 17% of its points dominated by other algorithms. In this case, the percentages of SPEA2 and our algorithm are: 18% and 26%, respectively. The Pareto fronts for this function are shown in Figure 3.12.

In general, in the first two functions our algorithm obtained very good results. The values of the measures on these two functions indicate that our algorithm was able to approximate the true Pareto front in each case as well as the other algorithms. This is reflected in the values of the SC measure, too. In the case

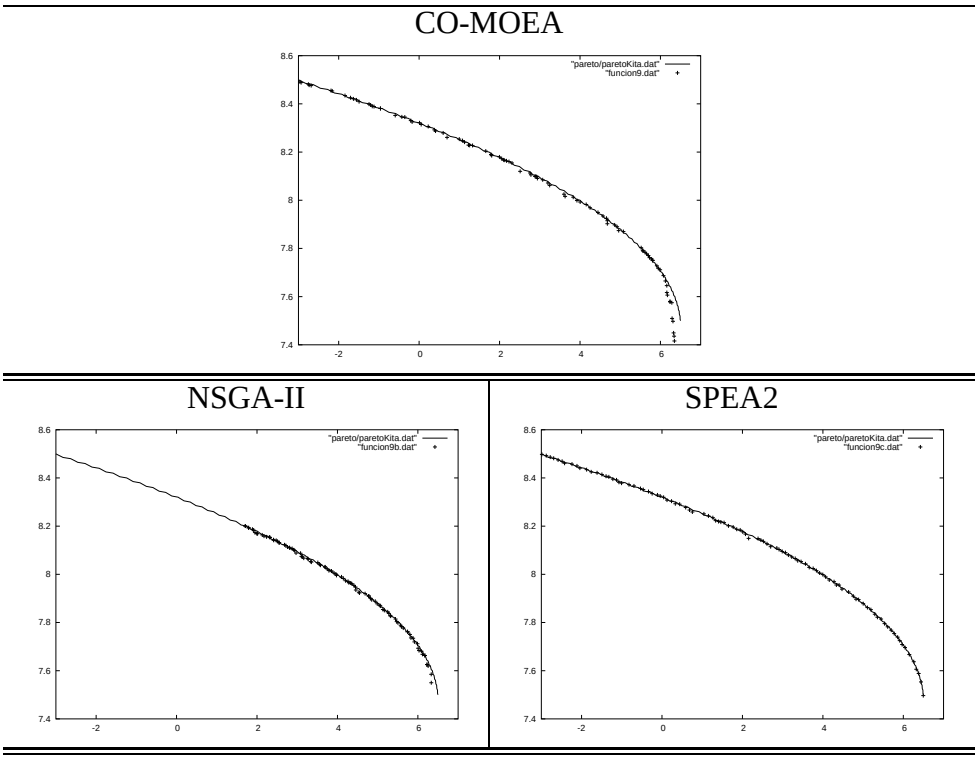


Figure 3.11: Pareto fronts obtained by our approach (CO-MOEA), the NSGA-II [27] and the SPEA2 [118], for test function Kita.

of the third and fourth functions, our algorithm obtained results somehow poor with respect to the other algorithms regarding the ER measure. This means that our algorithm couldn't find as many points of the true Pareto front as the other algorithms, and this is the reason for the relatively high values of our algorithm in the SC measure. However, the values obtained in these cases on the GD measure indicate that our algorithm was as close of the true Pareto front as the other algorithms. Regarding the distribution (SP measure), except in the fourth function, the values of our algorithm are not as good as those of the other algorithms.

Results using high-dimensional functions

With the aim of exploring the scalability capacities of our coevolutionary approach, we finally present some results using three different high-dimensional bi-objective functions: ZDT1, ZDT2 and ZDT3 (defined in Section 2.4.3, page

Test Function Tanaka				
		CO-MOEA	NSGA-II	SPEA2
ER	best	0.05	0.01	0.02
	median	0.16	0.08	0.06
	worst	0.29	0.17	0.10
	average	0.15	0.08	**0.06**
	std. dev.	0.0461	0.0339	0.0216
GD	best	0.0009	0.0008	0.0010
	median	0.0012	0.0013	0.0012
	worst	0.0015	0.0016	0.0014
	average	**0.0012**	**0.0012**	**0.0012**
	std. dev.	0.0001	0.0002	0.0001
SP	best	0.0047	0.0065	0.0037
	median	0.0085	0.0099	0.0052
	worst	0.0185	0.0155	0.0079
	average	0.0092	0.0101	**0.0053**
	std. dev.	0.0027	0.0022	0.0009

Two Set Coverage Metric SC				
	X	SC(X,I)	SC(X,II)	SC(X,II)
I	CO-MOEA	0.00	0.14	0.18
II	NSGA-II	0.17	0.00	0.17
III	SPEA2	0.34	0.20	0.00
Average		26%	**17%**	18%

Table 3.5: Comparison of results between our approach (denoted by CO-MOEA), the NSGA-II [27] and the SPEA2 [118] for test function Tanaka.

19). We performed 30 runs and a total of 20,000 evaluations (on average), for each test function. Table 3.6 shows the results obtained by our algorithm, with respect to the ER, GD and SP measures. Also, Figure 3.13 shows the solutions obtained by our coevolutionary approach, for the three functions. The plots on the left show the union of the 30 Pareto fronts obtained. The plots on the right show the nondominated solutions obtained from the union of the 30 Pareto fronts obtained. As we can see in Table 3.6, the results of the ER measure indicate that our algorithm was not able to converge to the true Pareto fronts of any of the three functions, that is, our algorithm was not able to obtain optimal solutions. On the other hand, although the results of the GD and SP measures indicate that our approach was able to obtain good approximations of the true Pareto fronts, we can

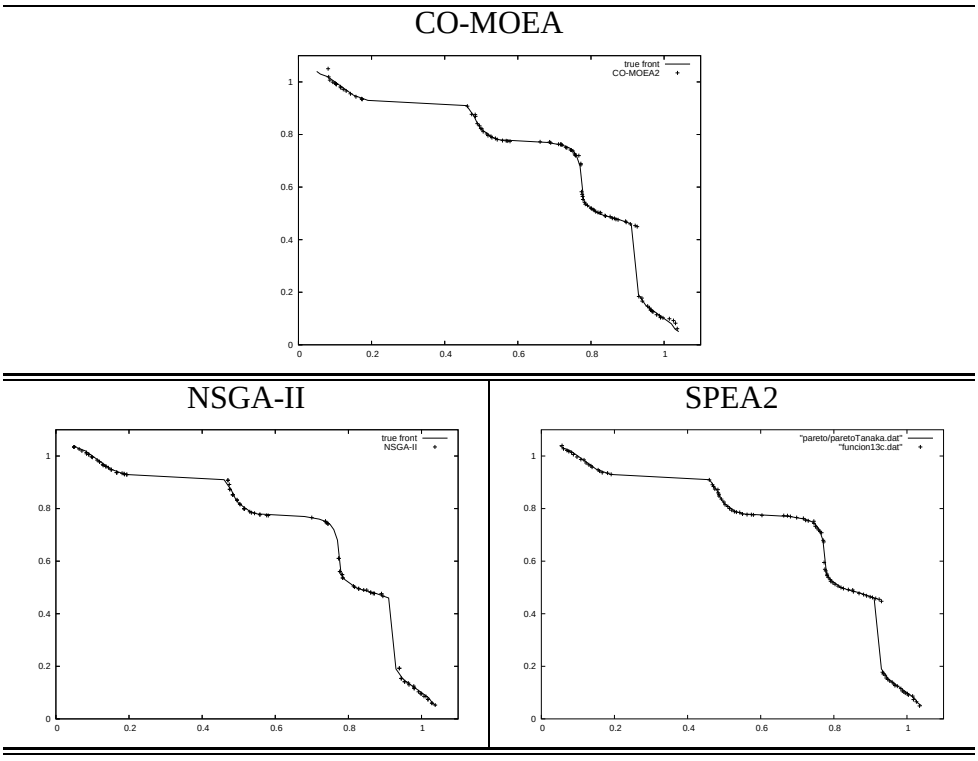


Figure 3.12: Pareto fronts obtained by our approach (CO-MOEA), the NSGA-II [27] and the SPEA2 [118], for test function Tanaka.

see in Figure 3.13 that the obtained sets of solutions don't cover the whole true Pareto fronts, in the majority of cases. In function ZDT1, our algorithm obtains a partial approximation of the true Pareto front. In function ZDT2, our algorithm collapses on just one point (near point (0,1)). Finally, it was in function ZDT3 where our algorithm obtained the best approximations.

In this case, we didn't compare our algorithm against any other MOEA, since it was not able to converge to the true Pareto fronts of the problems used.[2] Nevertheless, results obtained by other MOEAs on these test functions will be presented in the next chapter.

[2] As we will see in the next chapter, other MOEAs are able to converge to the true Pareto fronts of the problems used, performing the same number of function evaluations (20000).

		ZDT1	ZDT2	ZDT3
ER	best	0.82	0.40	1.00
	median	0.97	1.00	1.00
	worst	1.00	1.00	1.00
	average	0.95	0.98	1.00
	std. dev.	0.0475	0.1095	0.0000
GD	best	0.0026	0.00001	0.0041
	median	0.0061	0.0006	0.00114
	worst	0.0797	0.0025	0.0599
	average	0.0107	0.0008	0.0154
	std. dev.	0.0159	0.0006	0.0126
SP	best	0.0014	0.0000	0.0025
	median	0.0076	0.0000	0.0242
	worst	0.4150	0.0001	0.3462
	average	0.0326	0.00001	0.0606
	std. dev.	0.0854	0.00003	0.0910

Table 3.6: Results obtained by our approach in high-dimensional functions.

3.5 Conclusions

We presented a coevolutionary multi-objective evolutionary algorithm whose main idea is to detect the most "promising" sub-regions of the search space and focus the search on them. Thus, our approach is expected to ignore the useless subregions of the search space and, as a consequence, reduce computational cost. With this aim, the final version of the proposed algorithm applies a clustering algorithm on the set of decision variables of the known Pareto front. The proposed approach was validated using several test functions taken from the specialized literature. Our comparative study showed that the proposed approach provides competitive results, however, it was marginally outperformed by two other algorithms that are representative of the state-of-the-art in the area, especially regarding the distribution of the Pareto fronts produced.

On the other hand, after doing experiments with test functions with high-dimensional decision spaces, we could observe that our algorithm was not able to converge to the true Pareto fronts of such problems. Thus, since other MOEAs are able to obtain such optimal solutions, while performing the same number of function evaluations (as we will see in the next chapter), we concluded that some improvements to our algorithm were needed. That is, our coevolutionary approach

is expected to obtain Pareto fronts of at least the same quality of those obtained by other approaches, while performing the same number of functions evaluations. In fact, our proposed algorithm should be able to obtain the same results but performing a smaller number of function evaluations. For this reason, since our co-evolutionary scheme was using MOGA [36] as its search engine, we considered the idea of using a more efficient multi-objective algorithm in order to improve the obtained results. Thus, the following part of our work concerns the design of a new algorithm to be used as a search engine. In the following chapter, we will describe the new approach designed.

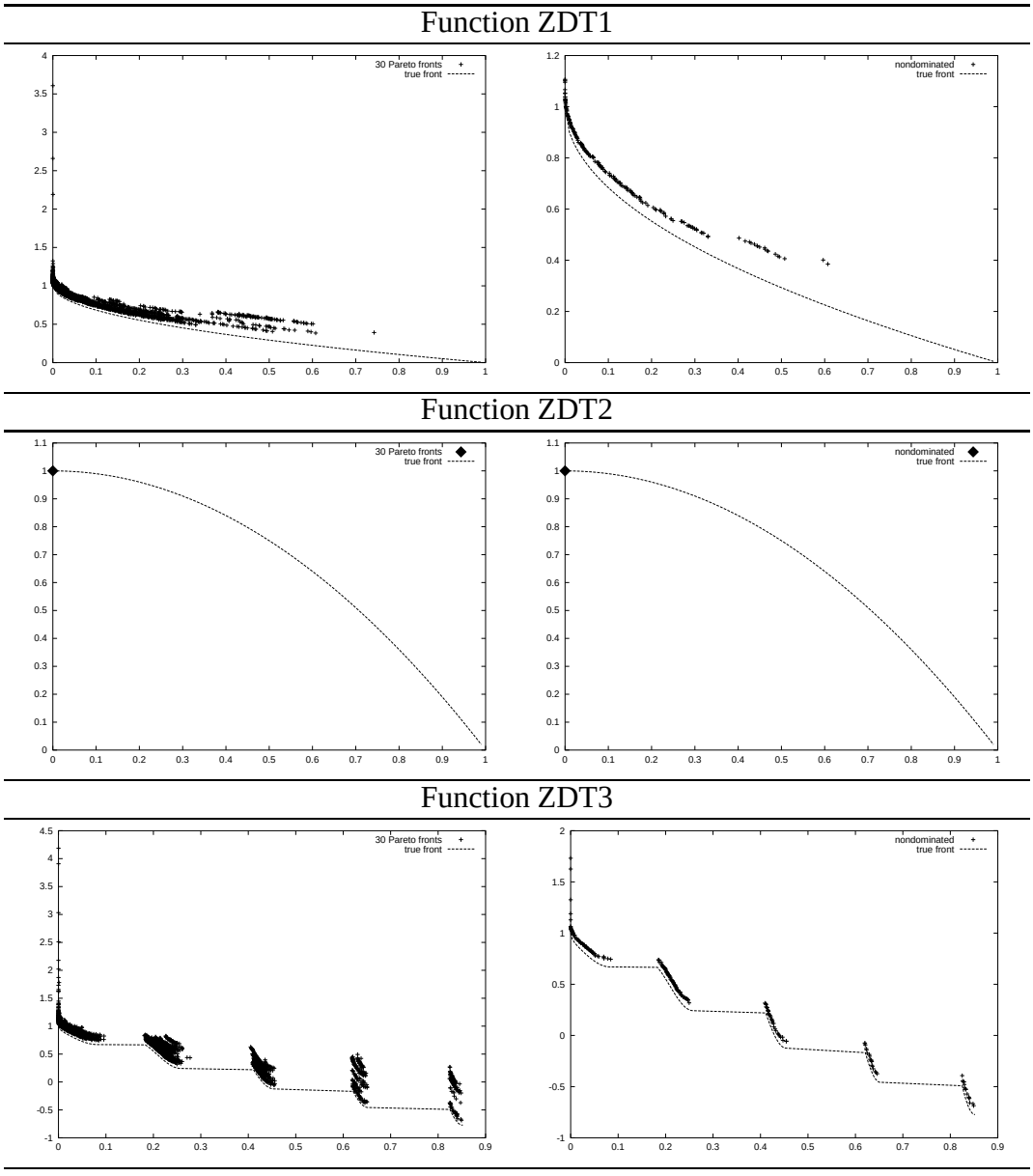


Figure 3.13: Solutions obtained by our coevolutionary approach, for functions ZDT1, ZDT2 and ZDT3. The plots on the left show the union of the 30 Pareto fronts obtained. The plots on the right show the nondominated solutions obtained from the union of the 30 Pareto fronts obtained.

Chapter 4

Multi-Objective Particle Swarm Optimization

In this chapter, we describe the new algorithm that we designed with the aim of using it as a search engine in the previously described coevolutionary approach.

4.1 Particle Swarm Optimization

Kennedy and Eberhart [55] initially proposed the swarm strategy for optimization. The Particle Swarm Optimization (PSO) algorithm is a population-based search algorithm based on the simulation of the social behavior of birds within a flock. In PSO, individuals, referred to as particles, are "flown" through a hyperdimensional search space. Changes to the position of the particles within the search space are based on the social-psychological tendency of individuals to emulate the success of other individuals.

A swarm consists of a set of particles, where each particle represents a potential solution. The position of each particle is changed according to its own experience and that of its neighbors. Let $\mathbf{x}_i(t)$ denote the position of particle i, at time step t. The position of particle i is then changed by adding a velocity $\mathbf{v}_i(t)$ to the current position, i.e.:

$$\mathbf{x}_i(t) = \mathbf{x}_i(t-1) + \mathbf{v}_i(t) \tag{4.1}$$

The velocity vector drives the optimization process and reflects the socially exchanged information. In general, based on the social information that is exchanged at each iteration, there are three different versions of the PSO algorithm:

- **Individual best.** Each particle compares its current position to its own best position (*pbest*), only.
- **Local best.** Particles are influenced by the best position within their neighborhood (*lbest*), as well as their own past experience (*pbest*).
- **Global best.** Each particle uses its history of experiences in terms of its own best solution thus far (*pbest*) but, in addition, the particle uses the position of the best particle from the entire swarm (*gbest*).

In the global best version (used here) of PSO, the velocity vector changes in the following way:

$$\mathbf{v}_i(t) = W\mathbf{v}_i(t-1) + C_1 r_1(\mathbf{x}_{pbest_i} - \mathbf{x}_i(t-1)) + C_2 r_2(\mathbf{x}_{gbest_i} - \mathbf{x}_i(t-1)) \quad (4.2)$$

where W is the inertia weight, C_1 and C_2 are the learning factors, and $r_1, r_2 \in [0,1]$ are random values. The inertia weight W is employed to control the impact of the previous history of velocities on the current velocity of a given particle. On the other hand, the learning factors represent the attraction that a particle has toward either its own success or that of its neighbors. C_1 is the *cognitive* learning factor and represents the attraction that a particle has toward its own success. C_2 is the *social* learning factor and represents the attraction that a particle has toward the success of its neighbors. Both, C_1 and C_2, are usually defined as constants.

Figure 4.1 shows the way in which the general PSO algorithm works. First, the swarm is initialized: positions and velocities. The corresponding *pbest* of each particle is initialized and the *gbest* is located. Then, for a maximum number of iterations, each particle flies through the search space updating its position (using (4.1) and (4.2)) and its *pbest* and, finally, the *gbest* solution is updated too.

The successful application of PSO in many single objective optimization problems reflects its effectiveness [56]. However, PSO must be modified in order to apply it to multi-objective problems. In most approaches (which will be generically called MOPSOs, for Multiple-Objective Particle Swarm Optimizers), the major modifications of the PSO algorithm are the selection process of *pbest* and *gbest* [20, 68, 114, 6].

4.2 Related Work

There have been several proposals to extend PSO to handle multiple objectives. We will review next the most representative of them:

```
Begin
    Initialize swarm
    Locate gbest
    g ← 0
    While g < gmax
            For each particle
                    Update Position (Flight)
                    Evaluation
                    Update pbest
            EndFor
            Update gbest
            g ← g + 1
    EndWhile
    Report gbest
End
```

Figure 4.1: Pseudocode of the general PSO algorithm.

- Ray and Liew [78]: This algorithm uses Pareto dominance and combines concepts of evolutionary techniques with the particle swarm. The approach uses crowding to maintain diversity (by means of a roulette selection scheme of leaders[1] based on this value) and a multilevel sieve to handle constraints (for this, the authors adopt the constraint and objective matrices proposed in some of their previous research [77]).

- Hu and Eberhart [47]: In this algorithm, only one objective is optimized at a time using a scheme similar to lexicographic ordering [21]. Lexicographic ordering tends to be useful only when few objective functions are used (two or three), and it may be sensitive to the ordering of the objectives. In further work, Hu et al. [48] adopted a secondary population (called "extended memory") and introduced some further improvements to their dynamic neighborhood PSO approach.

- Fieldsend and Singh [34]: This approach uses an unconstrained elite archive (in which a special data structure called "dominated tree" is adopted) to

[1] It is worth noting that the *leader* or *guide* is used as the *gbest* solution used in the formula for updating positions.

store the nondominated individuals found along the search process. The archive interacts with the primary population in order to define local guides. This approach also uses a "turbulence" operator that is basically a mutation operator that acts on the velocity value used by PSO.

- Coello et al. [17, 20]: This proposal is based on the idea of having a global repository in which every particle will deposit its flight experiences after each flight cycle. Additionally, the updates to the repository are performed considering a geographically- based system defined in terms of the objective function values of each individual; this repository is used by the particles to identify a leader that will guide the search. It also uses a mutation operator that acts both on the particles of the swarm, and on the range of each design variable of the problem to be solved. In more recent work, Toscano and Coello [103] use the concept of Pareto dominance to determine the flight direction of a particle. The authors adopt clustering techniques to divide the population of particles into several swarms in order to have a better distribution of solutions in decision variable space. In each sub-swarm, a PSO algorithm is executed and, at some point, the different sub-swarms exchange information: the leaders of each swarm are migrated to a different swarm in order to variate the selection pressure. Also, this approach does not use an external population since elitism in this case is an emergent process derived from the migration of leaders.

- Mostaghim and Teich [68]: They propose a sigma method in which the best local guides for each particle are adopted to improve the convergence and diversity of a PSO approach used for multi-objective optimization. They also use a "turbulence" operator, but applied on decision variable space. The idea of the sigma method is similar to compromise programming. The use of the sigma values increases the selection pressure of PSO (which was already high). This may cause premature convergence in some cases. In further work, Mostaghim and Teich [67] study the influence of ε-dominance [58] on MOPSO methods. ε-dominance is compared with existing clustering techniques for fixing the archive size and the solutions are compared in terms of computational time, convergence and diversity. The results show that the ε-dominance method can find solutions much faster than the clustering technique with a comparable (and even better in some cases) convergence and diversity. The authors suggest a new diversity measure (sigma method) inspired on their previous work [68]. Also, based on the idea that

the initial archive from which the particles have to select a local guide has influence on the diversity of solutions, the authors propose the use of successive improvements using a previous archive of solutions. In more recent work, Mostaghim and Teich [69] propose a new method called *covering*MOPSO (cvMOPSO). This method works in two phases. In phase 1, a MOPSO algorithm is run with a restricted archive size and the goal is to obtain a good approximation of the Pareto-front. In the phase 2, the nondominated solutions obtained from the phase 1 are considered as the input archive of the cvMOPSO. The particles in the population of the cvMOPSO are divided into subswarms around each nondominated solution after the first generation. The task of the subswarms is to cover the gaps between the nondominated solutions obtained from the phase 1. No restrictions on the archive are made in phase 2.

- Li [59]: This approach incorporates the main mechanisms of the NSGA-II [27] to the PSO algorithm. It combines the population of particles and all the personal best positions of each particle, and selects the best particles among them to conform the next population. It also selects the leaders randomly from the leaders set among the best of them, based on two different mechanisms: using a niche count and using a crowding distance. In more recent work, Li [60] proposes the *maximinPSO*, which uses a fitness function derived from the maximin strategy [3] to determine Pareto-domination. The author shows that one advantage of this approach is that no additional clustering or niching technique is needed, since the maximin fitness of a solution can tell us not only if a solution is dominated or not, but also if it is clustered with other solutions, i.e., the approach also provides diversity information.

- Baumgartner et al. [7]: This approach uses weighted sums (i.e., linear aggregating functions). In this approach, the swarm is equally partitioned into n subswarms, each of which uses a different set of weights and evolves into the direction of its own swarm leader. The approach adopts a gradient technique to identify the Pareto optimal solutions.

- Chow and Tsui [14]: In this paper, a novel autonomous agent response learning algorithm is presented. The authors propose to decompose the award function into a set of local award functions and, in this way, to model the response extraction process as a multi-objective optimization problem. A modified PSO called “Multi-Species PSO” is introduced by considering

each objective function as a species swarm. A communication channel is established between the neighboring swarms for transmitting the information of the best particles, in order to provide guidance for improving their objective values. Also, the authors propose to modify the equation used to update the velocity of each particle, considering also the global best particle of its neighboring species.

- Alvarez-Benitez et al. [1]: They propose methods based exclusively on Pareto dominance for selecting guides from a unconstrained nondominated archive. Three different techniques are presented: *Rounds* which explicitly promotes diversity, *Random* which promotes convergence and *Prob* which is a weighted probabilistic method and forms a compromise between *Random* and *Rounds*. Also, the authors propose and evaluate four mechanisms for confining particles to the feasible region, that is, constraint-handling methods. The authors show that probabilistic selection favoring archival particles that dominate few particles provides good convergence towards and coverage of the Pareto front. Also, they concluded that allowing particles to explore regions close to the constraint boundaries is important to ensure convergence to the Pareto front.

- Ho et al. [44]: The authors propose a novel formula for updating velocity and position particles, based on three main modifications to the known flight formula for the *gbest* version of PSO. First, since the authors argue that the random factors r_1 and r_2 in Equation (4.2) are not completely independent, they propose to use: $r_2 = 1 - r_1$. Second, they propose to incorporate the term $(1-W)$ in the second and third terms of Equation (4.2), where $W = rnd(0,1)$. Third (and last), under the argument of allowing a particle to fly sometimes back, the authors propose to allow the first term of Equation (4.2) being negative with a 50% probability. On the other hand, the authors introduce a "craziness" operator in order to promote diversity within the swarm. This "craziness" operator is applied (with certain probability) to the velocity vector before updating the position of a particle. Finally, the authors introduce one external archive for each particle and one global external archive for the whole swarm. The archive of each particle stores the latest Pareto solutions found by the particle and the global archive stores the current Pareto optimal set. Every time a particle updates its position, it selects its personal best from its own archive and the global best from the global archive. In both cases, the authors use a roulette selection mechanism

based on the fitness values of the particles (assigned using the mechanism originally proposed by Zitzler et al. [119], for the SPEA algorithm) and on an "age" variable that the authors introduce and that is increased at each generation.

- Villalobos-Arias et al. [108]: The authors propose a new mechanism to promote diversity in multi-objective optimization problems. Although the approach is independent of the search engine adopted, they incorporate it into the MOPSO proposed in [20]. The new approach is based on the use of stripes that are applied on the objective function space. Based on an analysis for a bi-objective problem, the main idea of the approach is that the Pareto front of the problem is "similar" to the line determined by the minimal points of the objective functions. In this way, several points (that the authors call stripe centers) are distributed uniformly along such line, and the particles of the swarm are assigned to the nearest stripe center. When using this approach for solving multi-objective problems with PSO, one leader is used in each stripe. Such leader is selected minimizing a weighted sum of the minimal points of the objective functions. The authors show that their approach overcomes the drawbacks on other popular mechanisms such as ε-dominance [58] and the sigma method proposed in [68].

- Salazar-Lechuga and Rowe [89]: The main idea of this approach is to use PSO to guide the search with the help of niche counts (applied on objective function space) [41] to spread the particles along the Pareto front. The approach uses an external archive to store the best particles (nondominated particles) found by the algorithm. Since this external archive helps to guide the search, the niche count is calculated for each of the particles in the archive and the leaders are chosen from this set by means of an stochastic sampling method (roulette wheel). Also, the niche count is used as a criterion to update the external archive. Each time the archive is full and a new particle wants to get in, its niche count is compared with the niche count of the worst solution of the archive. If the new particle is better than the worst particle, then the new particle enters into the archive and the worst particle is deleted. Niche counts are updated when inserting or deleting a particle from the archive.

- Janson and Merkle [52]: The authors propose a hybrid particle swarm optimization algorithm for multi-objective optimization, called ClustMPSO.

ClustMPSO combines the PSO algorithm with clustering techniques to divide all particles into several subswarms. For this aim, the authors use the *K*-means algorithm. Each subswarm has its own nondominated front and the total nondominated front is obtained from the union of the fronts of all the subswarms. Each particle randomly selects its neighborhood best (*lbest*) particle from the nondominated front of the swarm to which it belongs. Also, a particle only selects a new *lbest* particle when the current is no longer a nondominated solution. On the other hand, the personal best (*pbest*) of each particle is updated based on dominance relations. Finally, the authors define that a subswarm is dominated when none of its particles belongs to the total nondominated front. In this way, when a subswarm is dominated for a certain number of consecutive generations, the subswarm is relocated. The proposed algorithm is tested on an artificial multi-objective optimization function and on a real-world problem from biochemistry, called the molecular docking problem. The authors reformulate the molecular docking problem as a multi-objective optimization problem and, in this case, the updating of the *pbest* particle is also based on the weighted sum of the objectives of the problem. ClustMPSO outperforms a well-known Lamarckian Genetic Algorithm that had been previously adopted to solve such problem.

- Xiao-hua et al. [113]: The authors propose an Intelligent Particle Swarm Optimization (IPSO) algorithm for multi-objective problems based on an Agent-Environment-Rules (AER) model to provide an appropriate selection pressure to propel the swarm population towards the Pareto optimal front. In this model, the authors modify the *global best* flight formula including the *lbest* position of the neighborhood of each particle. The neighborhood of a particle is determined by a lattice-like topology. On the other hand, each particle is taken as an agent particle with the ability of memory, communication, response, cooperation and self-learning. Each particle has its position, velocity and energy, which is related to its fitness. All particles live in a latticelike environment, which is called an agent lattice, and each particle is fixed on a lattice-point. In order to survive in the system, they compete or cooperate with their neighbors so that they can gain more resources (increase energies). Each particle has the ability of cloning itself, and the number of clones produced depends of the energy of the particle. General agent particles and latency agent particles (those who have smaller energy but contain certain features—e.g., favoring diversity—that make them good

candidates to be cloned) will be cloned. The aim of the clonal operator (which is modeled in the clonal selection theory also adopted with artificial immune systems [70]) is to increase the competition among particles, maintain diversity of the swarm and improving the convergence of the process. Also, a clonal mutation operator is used. Leaders are selected based on the energy values of the particles. Finally, this approach adopts an external archive in order to store the nondominated solutions found throughout the run and to provide the final solution set.

Although some of the approaches previously described represent very illustrative proposals (like the unconstrained external archive in [34] and the study of different leader selection techniques in [1]), some others provide proposals with clear disadvantages. For example, some approaches require information of the problem that is not always available (like the gradient in [7]) or implementations that can be considered somehow complicated (like the species per function in [14], the clustering techniques in [52, 20] and the intelligent system in [113]). Also, some other approaches require specific conditions of the objective functions that imply some previous management of the problem (like the nonnegative condition in [68]), or include parameters which are very important for the behavior of the algorithm and that have to be fixed by the user (like the σ_{share} in [89]). In fact, some approaches don't seem to be scalable for solving problems with more than two objectives (like the stripes proposal in [108]).

In general, among the most important features of the MOPSO approaches are the mechanisms used to define, select and maintain the available set of leaders. As we could see, the majority of the approaches previously proposed maintain a set of leaders that grows at each iteration, that is, the size of such set is unbounded. This characteristic introduces some difficulties when a leader has to be selected for updating the position of one particle (as in the case of the parameteres needed for that sake in [59, 60]) or when a size fixed set of solutions has to be returned as the output of the algorithm (like in [59, 60, 103], where the proposed approaches don't have any mechanism to fix the final set of solutions).

In the next section, we describe a new proposal of our MOPSO approach, which does not require any knowledge or specific condition of the problem. Also, it is relatively easy to implement and needs very few parameters which are also very easy to set. On the other hand, the proposed algorithm incorporates mechanisms for maintaining fixed the size of the available set of leaders and that allow the design of easy techniques for selecting leaders at each iteration, and also for selecting the final solutions of the algorithm.

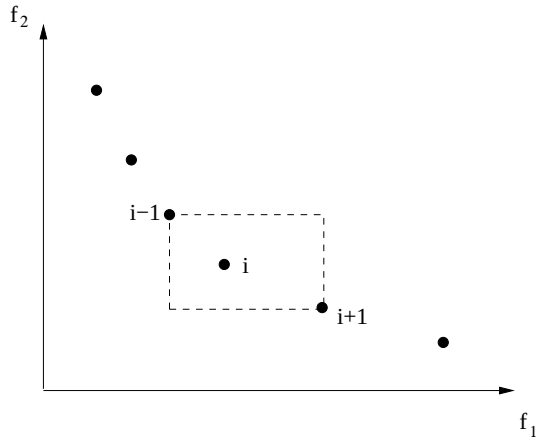


Figure 4.2: The crowding factor gives us an idea of how crowded are the closest neighbors of a given particle, in objective function space.

4.3 PSO-Based Multi-Objective Approach

It should be obvious that the main issue when extending PSO to deal with multiple objectives is how to generalize the concept of leader in the presence of several (equally good) solutions. The most straightforward approach is simply to consider every nondominated solution as a new leader. This approach has, however, the drawback of increasing the size of the set of leaders very quickly. This increase on the size of the leaders set is very important because such set has to be updated at each generation, and this can become a very expensive process. Also, the size of the set of leaders has a significant impact on the selection of a leader in order to make a movement through the search space at each generation.

If the set of leaders becomes too large, the election of a good leader turns out to be difficult (how do we discriminate from among many nondominated individuals?). This particular issue has not been addressed in most of the research conducted in this area and is the main focus of this work. In our approach, we use a crowding factor [27] in order to establish a second discrimination criterion, additional to Pareto dominance. See Figure 4.2.

Since leader selection is a key component in PSO, for each particle, we select the leader using a combination of two different techniques:

I Given a particle x, we assign as its leader a particle y, randomly chosen, but *if and only if* y **dominates** x. If it is the case that x is a nondominated

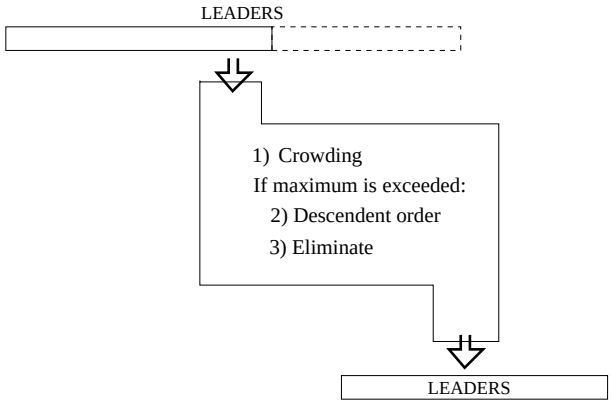


Figure 4.3: Procedure followed to maintain fixed the size of the set of leaders.

particle, the leader will be randomly chosen.

II A binary tournament based on the crowding value of the leaders.

Technique I is used in order to speed up the convergence to the true Pareto front, and technique II is used in order to avoid losing diversity within the swarm. After an extensive series of experiments, we determined that our approach reached its best performance when using technique I during 97% of the time and technique II the remaining 3% of the time.

The crowding criterion is also adopted to decide what leaders to keep over generations when the maximum list size has been exceeded. The maximum size of the set of leaders is fixed equal to the size of the swarm or population (this value is provided by the user as a parameter). After each generation, the set of leaders is updated, and so are the corresponding crowding values. If the size of the set of leaders is greater than the maximum allowable size, only the best leaders are retained based on their crowding value. The rest of the leaders are eliminated. This process is illustrated in Figure 4.3. Although there are previous approaches that use the crowding factor to select the leaders (see for example [78, 59]), our approach is the first to adopt this information to fix the size of the set of leaders. This feature of our algorithm considerably simplifies the mechanism to control the set of leaders without requiring any additional parameter or selection criterion (e.g., in [59], a parameter is required to define the portion of the list of leaders to be retained). Additionally, the use of this selection criterion promotes diversity within the swarm.

When adopting PSO for solving multi-objective optimization problems, the use of a mutation operator is very important in order to escape from local optima and to improve the exploratory capabilities of PSO which, in some cases, becomes severely limited (see for example [20]). Several researchers have used mutation operators before. For example, Fieldsend & Singh [34], Coello & Lechuga [17] and Mostaghim & Teich [68]. It is worth noticing that all of the previous proposals require of some user-defined parameter.

From our perspective, the election of a good mutation operator is a difficult task that has a significant impact on performance. Thus, in this work we propose the use of two mutation operators that are well-known in the EA literature: uniform mutation (i.e., the variability range of each decision variable is kept constant over generations) [39] and non-uniform mutation (i.e., the variability range of each decision variable decreases over time) [63]. Both operators act on the decision variables of the updated particle. These operators modify the values of the decision variables of a particle with a certain probability. This makes a significant difference with respect to the previous proposals in which all the decision variables are modified when the turbulence (or mutation) operator is applied. Additionally, we considered the possibility of not using mutation at all, since in some of our previous research we found that such condition may be beneficial in some cases [20].

Given the uncertainty regarding how much mutation to apply, we propose a scheme by which the swarm is subdivided into three parts (of equal size). Each sub-part of the swarm will adopt a different mutation scheme: the first sub-part will have no mutation at all, the second sub-part will have uniform mutation and the third sub-part will have non-uniform mutation. This process is illustrated in Figure 4.4. With the use of these different operators we are aiming to have the ability of exploring (uniform mutation) and exploiting (non-uniform mutation) the search space as the process progresses. The available set of leaders is the same for each of these sub-parts. Additionally, each particle can use as a leader a particle produced by a different sub-part of the swarm. In this way, the three different sub-parts of the swarm will share their particular success and the final results will be a combination of using different behaviors inside the same swarm.

In order to avoid the definition of extra parameters for the mutation operators, we adopt a rule of thumb commonly used in the EA literature: the mutation rate is defined as $1/codesize$, where *codesize* refers to the total length of the string that encodes all the decision variables of the problem. Since real-numbers encoding is adopted, in our case *codesize* is equal to the number of decision variables of each problem.

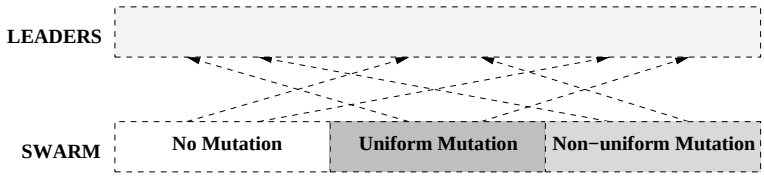


Figure 4.4: Graphical illustration of the subdivision of the swarm adopted by our scheme.

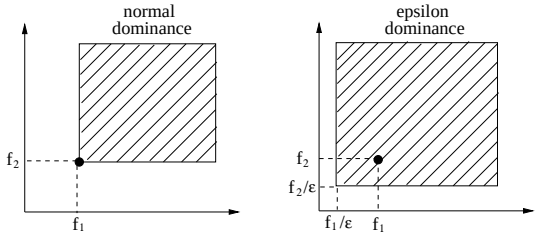


Figure 4.5: Graphical illustration of ε-dominance.

Finally, we adopt the concept of ε-dominance [58] in order to fix the size of the external archive that contains the (nondominated) solutions that will be reported by the algorithm. A decision vector x_1 is said to ε-dominate a decision vector x_2 for some $\varepsilon > 0$ iff: $f_i(x_1)/\varepsilon \leq f_i(x_2), \forall i = 1,...,m$ and $f_i(x_1)/\varepsilon < f_i(x_2)$, for at least one $i = 1,...,m$. See Figure 4.5. It is worth noting that, when using ε-dominance, the size of the final external archive depends on the ε-value, which is normally a user-defined parameter [58]. For the sake of simplicity, we consider the same value of ε for all the objective functions.

Figure 4.6 shows the way in which our algorithm works. We have marked with *italics* the processes that make this algorithm different from the general PSO algorithm. First, we initialize the swarm. The nondominated particles found in the swarm will be introduced into the set of leaders. Later on, the crowding factor of each leader is calculated. At each generation, for each particle, we perform the flight and apply the corresponding mutation operator based on the subdivision of the swarm previously described. In order to perform the flight of each particle, the changes to the velocity vector are done in the following way:

$$\mathbf{v}_i(t) = W\mathbf{v}_i(t-1) + C_1 r_1(\mathbf{x}_{pbest_i} - \mathbf{x}_i(t-1)) + C_2 r_2(\mathbf{x}_{gbest_i} - \mathbf{x}_i(t-1))$$

where $W = random(0.1, 0.5)$, $C_1, C_2 = random(1.5, 2.0)$, and $r_1, r_2 = random$

```
Begin
    Initialize swarm
    Locate leaders
    Send leaders to ε-archive
    crowding(leaders)
    g ← 0
    While g < gmax
            For each particle
                    Select leader, Flight
                    Mutation
                    Evaluation, Update pbest
            EndFor
            Update leaders
            Send leaders to ε-archive
            crowding(leaders)
            g ← g + 1
    EndWhile
    Report results in ε-archive
End
```

Figure 4.6: Pseudocode of our algorithm.

$(0.0, 1.0)$. Note that most of the previous PSO proposals fix the values of W, C_1 and C_2 instead of using random values as in our case. The only exception that we know (in the specific case of MOPSOs) is some of our own previous work [103]. We adopted this scheme since we found it as a more convenient way of dealing with the difficulties of fine tuning the parameters W, C_1 and C_2 for each specific test function.

Then, we proceed to evaluate the particle and update its personal best position (*pbest*). A new particle replaces its *pbest* position if the current *pbest* is dominated by the new particle or if both are incomparable (i.e., they are both non-dominated with respect to each other). After all the particles have been updated, the set of leaders is updated, too. Obviously, only the particles that outperform their *pbest* position will try to enter the leaders set. Once the leaders set has been updated, the ε-archive is updated. Finally, we proceed to update the crowding values of the set of leaders and we eliminate as many leaders as necessary in order to avoid exceeding the allowable size of the leaders set. This process is repeated a fixed number (*gmax*) of iterations.

The parameters needed by our approach are:

1. *swarmsize*: size of the swarm (defined by the user).
2. *gmax*: number of iterations (defined by the user).
3. *pm*: bit mutation probability (fixed by the algorithm).
4. ε: value for the bounding the size of the ε-archive (defined by the user).

4.4 Discussion of Results

Previous to the final version of our algorithm, described in the previous section, we implemented a first version whose only difference was that it only adopted technique I as its selection mechanism. The results of this previous version can be seen in [95] and [98].

In this section, we discuss the results obtained by the final version of our approach [97]. All the corresponding tables and figures can be consulted in Appendix A, page 157. To validate our approach, we performed both quantitative (adopting four performance measures) and qualitative comparisons (plotting the Pareto fronts produced) with respect to two MOEAs that are representative of the state-of-the-art in the area: the Strength Pareto Evolutionary Algorithm 2 (SPEA2) [118], and the Nondominated Sorting Genetic Algorithm II (NSGA-II) [27]. We also compared our approach against three PSO-based approaches recently proposed: MOPSO [21], Sigma-MOPSO [68] and Cluster-MOPSO [103]. For our comparative study, we implemented two unary and two binary measures of performance: Success Counting (SCC), Inverted Generational Distance (IGD), Two Set Coverage (SC) and the Two Set Difference Hypervolume (HV). All these measures are defined in Section 2.4.4, page 25.

For each of the test functions used, we performed 20 runs per algorithm. All the algorithms compared adopted real-numbers encoding. The parameters of each approach were set such that they all performed 20000 objective function evaluations.

It is very important to mention that our approach was firstly tested with functions with low dimensionality and the results were very satisfactory. In this way, we proceeded to test our approach with high dimensional functions and the corresponding results are shown here. Also, we would like to mention that, given the limitations imposed by the No Free Lunch Theorem [112], we do not expect to obtain conclusive (i.e. general) results, but only partial results about the behavior of our algorithm. The test functions adopted in our comparative study were: ZDT1,

ZDT2, ZDT3, ZDT4, DTLZ2, DTLZ4 and DTLZ6 (defined in Section 2.4.3, page 19).

The PSO approaches will be identified with the following labels: MOPSO refers to the approach reported in [17, 20], sMOPSO refers to the approach reported in [68], cMOPSO refers to the approach reported in [103], and OMOPSO (Our Multi-Objective Particle Swarm Optimizer) refers to our approach.

The Pareto fronts that we will show correspond to the nondominated vectors obtained from the union of the 20 obtained Pareto fronts. It should be noted that the Pareto fronts shown were also used to apply the binary measures of performance. All the algorithms, except for the Cluster-MOPSO, were set such that they provided Pareto fronts with 100 points. The Cluster-MOPSO does not have a scheme to fix the size of its final archive. Thus, in order to allow a fair comparison with respect to the Cluster-MOPSO, the values of the SCC measures were scaled to the interval [0,100].

From Table A.1 to Table A.14 we show the values of the performance measures obtained for each of the algorithms compared.

Through the use of binary measures of performance, and under certain conditions, we can conclude that a certain algorithm is better than another [120]. In this work, since we use two different binary measures, we will conclude that an algorithm is better than another when at least one of the measures indicates so, according to the definitions previously provided (see Section 2.4.4, page 25). Since the conditions to conclude that an algorithm is better that another using the binary measures are very difficult to satisfy in most cases, we will use the values obtained by the SC binary measure in order to conclude *partial* results: We will say that an algorithm A is *relatively* better than algorithm B when SC(A,B)¿SC(B,A), and *almost* better than B when SC$(B,A) = 0$ and SC$(A,B) > 0.9$. The values of the HV binary measure will be used to make only conclusions of the type: algorithm A is better than algorithm B, just like it was previously defined.

Function ZDT1. From Table A.1, we can conclude that our algorithm obtained the best results with respect to the SCC measure and, with respect to the IGD measure, we obtained results as good as those obtained by all the other MOEAs compared, improving the results obtained by the other three PSO-based approaches.

Regarding the binary measures (Table A.2) and considering both of them, we can conclude that OMOPSO is *relatively* better than the rest of the algorithms, except for sMOPSO, since sMOPSO is *relatively* better than OMOPSO. In fact, OMOPSO is *almost* better than MOPSO and cMOPSO.

We will now analyze in more detail the results obtained by our algorithm in

these measures. We cannot conclude that OMOPSO is better than the NSGA-II (for example) since SC(OMOPSO,NSGA-II)$\neq$ 1 but, since SC(OMOPSO,NSGA-II)¿SC(NSGA-II,OMOPSO), OMOPSO is *relatively* better than NSGA-II. On the other hand, we have SC(MOPSO,OMOPSO)= 0 and SC(OMOPSO,MOPSO)=0.96, so OMOPSO is *almost* better than MOPSO. Although it should be clear that OMOPSO is better than MOPSO, the results obtained do not allow to reach this conclusion since OMOPSO lost the extreme superior point of the front, as we can see in Figure A.1. This is due to the use of the ε-dominance scheme to fix the number of solutions in the external archive. This also explains the positive values obtained for the binary hypervolume measure in the column of OMOPSO in Table A.2. In Figure A.2, we show the Pareto front obtained from the union of the MOPSO and OMOPSO fronts. We can see that the hypervolume corresponding to the front shown in Figure A.2 is marginally bigger than the hypervolume corresponding to the front of OMOPSO, giving a positive value to the difference in the binary measure. This exemplifies the sort of anomalous behavior that can go undetected even when using binary performance measures.

Function ZDT2. From Table A.3, we can conclude that our algorithm (OMOPSO) obtained the best results in both unary measures, with the largest number of points (on average) belonging to the true Pareto front and the minimum IGD (on average).

Regarding the binary measures (considering both of them) (Table A.4), we can conclude that OMOPSO is better than MOPSO and sMOPSO. Also, we can say that OMOPSO is *almost* better than NSGA-II and SPEA2, and it is *relatively* better than cMOPSO.

In this case, there are two interesting issues to discuss. First, we can see in the SC binary measure values that almost 80% of the points of the cMOPSO algorithm are concentrated on the top part of the true Pareto front. Thus, although the major part of the observed front of the cMOPSO algorithm (see Figure A.3) is not on the true Pareto front, the corresponding results on the SC measure are not as expected. Second, the sMOPSO algorithm obtained just one point: (0.0,1.0). It is very interesting to note that none of the other algorithms was able to generate this point, as we can see in the SC measure values from Table A.4. Fortunately, these problems with the SC measure are overcome by the HV measure with a small modification: the values that we have marked with an asterisk (*) in Table A.4 were originally positive. However, we changed them to correspond more closely with reality, since the hypervolume corresponding to the front of sMOPSO is zero.

Function ZDT3. From the results shown in Table A.5, we can conclude that our algorithm (OMOPSO) obtained the best results in both unary measures, with the largest number of points (on average) belonging to the true Pareto front and the minimum IGD (on average).

Regarding the binary measures (see Table A.6), we can conclude that OMOPSO is *relatively* better than the rest of the algorithms. In fact, OMOPSO is *almost* better than MOPSO and cMOPSO. We can see the Pareto fronts obtained for this function in Figure A.4.

Function ZDT4. Based on the results shown in Table A.7, we can conclude that our algorithm (OMOPSO) obtained the best results with respect to the two unary measures adopted, with the largest number of points (on average) belonging to the true Pareto front and the minimum IGD (on average).

Regarding the binary measures and considering both of them (see Table A.8), we can conclude that OMOPSO is better than the other three PSO-based approaches. Also, OMOPSO is *almost* better than NSGA-II and SPEA2.

In this case, our approach is only *almost* better than the NSGA-II and SPEA2 for the same reason that we discussed in the case of function ZDT1. OMOPSO lost the top extreme point of the Pareto front due to the use of the ε-dominance scheme. For this reason, OMOPSO can't dominate completely the fronts produced by NSGA-II and SPEA2. In fact, it can't even dominate the isolated points obtained by the other PSO-based approaches. Additionally, for this same reason we find positive values in the column of OMOPSO for the binary hypervolume measure. However, the binary hypervolume measure lead us to conclude the superiority of OMOPSO compared with the other PSO-based approaches.

It is very important to note that our algorithm was the only PSO-based approach that was able to generate the entire Pareto front of this function. This illustrates the effectiveness of the mechanisms adopted in our approach to maintain diversity and to select and filter out leaders. We can see the Pareto fronts obtained for this function in Figure A.5.

Function DTLZ2. From Table A.9, we can conclude that the MOPSO algorithm obtained the best results in this function with respect to the SCC measure with an average of 89 points belonging to the true Pareto front. However, as we can see in Figure A.6, all the points obtained by the MOPSO algorithm are concentrated on one of the inferior corners of the true Pareto front. This fact is reflected by the values obtained by the MOPSO algorithm in the IGD measure, giving the worst values in this case. In this case, although in the SCC measure our proposed ap-

proach didn't obtain better results compared with the other PSO-based approaches (except cMOPSO), the corresponding values of the IGD measure indicate that our algorithm obtained as good approximations to the true Pareto front as the other approaches.

Regarding the binary measures and considering both of them (see Table A.10), we can conclude that no algorithm was better than any other. Also, we can say that the OMOPSO is *relatively* better than the cMOPSO and the NSGA-II algorithms. As in function ZDT2 and ZDT4, the sign of the values that we have marked with an asterisk (*) in Table A.10 was changed to correspond more closely with reality, since the hypervolume corresponding to the front of MOPSO is less than the hypervolume of the other algorithms, but this fact is due to the poor distribution of the solutions obtained by MOPSO.

Function DTLZ4. From the results shown in Table A.11, we can conclude that sMOPSO obtained the best result in the SCC measure and almost the same quality (on average) than the best result in the IGD measure (obtained in this case by the SPEA2). In this case, our algorithm didn't obtain good results, in fact, it was able to improve only the results of the cMOPSO algorithm. However, as we can see in Figure A.7, our algorithm had a very similar behavior to the sMOPSO algorithm (which, in general, had the best performance in this function).

Regarding the binary measures and considering both of them (see Table A.12), we can conclude that all the algorithms are better than cMOPSO. Also, we can say that OMOPSO is *relatively* better than NSGA-II, MOPSO and sMOPSO. The values marked with an asterisk (*) in Table A.12 were changed for reasons similar to those provided in function DTLZ2.

Function DTLZ6. From Table A.13, we can conclude that our algorithm (OMOPSO) obtained the best results in the SCC measure and almost the same quality (on average) than the best result in the IGD measure (obtained in this case by the SPEA2).

Regarding the binary measures and considering both of them (see Table A.14), we can conclude that OMOPSO is better than sMOPSO. Also, OMOPSO is *relatively* better than NSGA-II, SPEA2, MOPSO and cMOPSO. The values marked with an asterisk (*) in Table A.14 were changed to correspond more closely with reality, since the hypervolume corresponding to the front of sMOPSO is zero. We can see the Pareto fronts obtained for this function in Figure A.8.

Overall Discussion. With respect to the unary performance measures, our ap-

proach obtained the best results in both measures in three functions (ZDT2, ZDT3 and ZDT4), the best results in one measure in other three functions (ZDT1, DTLZ2 and DTLZ6), and it was outperformed in both measures only in function DTLZ4.

In function DTLZ2, our algorithm was outperformed with respect to the SCC measure. However, in this function, our approach obtained as good results as the best results obtained, with respect to the IGD measure. On the other hand, similar results were obtained in functions ZDT1 and DTLZ6, where our approach was marginally outperformed with respect to the IGD measure. This indicates that OMOPSO was able to obtain a good approximation and a good number of points of the true Pareto fronts of six of the seven test functions used.

Regarding the binary measures, our approach was *relatively* outperformed only in three (out of seven) functions (ZDT1, DTLZ2 and DTLZ4). However, OMOPSO was at least *relatively* better than almost all the algorithms in all functions, except in function DTLZ2.

In this way, except in function DTLZ4, OMOPSO was clearly superior compared with the other PSO-based approaches adopted in our comparative study. Also, the results obtained by OMOPSO showed that it is highly competitive with respect to both NSGA-II and SPEA2, which are two algorithms representative of the state-of-the-art in evolutionary multi-objective optimization.

4.5 Impact of the Parameters of Our Approach

With the aim of exploring which parameters were the most important for the performance of our approach, we performed an Analysis of Variance (ANOVA).

As we mentioned before, the selection of a leader in our MOPSO approach is done by means of a combination of two different techniques: (I) by dominance and (II) by crowding values. In this way, our approach chooses technique I with certain probability (that we will call *Ps*), otherwise, it chooses technique II. After an extensive series of experiments, we fixed the value of *Ps* to 0.97. However, *Ps* is itself a parameter of our approach.

On the other hand, in order to perform the flight of each particle, the changes to the velocity vector are done using Equations (4.1) and (4.2), where we use: $W = random(0.1, 0.5)$, $C_1, C_2 = random(1.5, 2.0)$, and $r_1, r_2 = random\ (0.0, 1.0)$. However, W, C_1 and C_2 can be considered parameters of our approach.

Finally, although the value of ε is a parameter of our approach that determines the number of solutions provided by the algorithm, ε has some properties that

allowed us to design a relatively simple mechanism to adapt its value. This mechanism will be described in Section 4.6.1.

Thus, the parameters of our approach that were be considered in our study are: *swarmsize* (size of the swarm), *gmax* (number of iterations), *Ps* (selection probability) and W, C_1 and C_2 (velocity update formula). For such study, we considered 3 different levels for each of the parameters of our approach:

- *swarmsize*: 50, 100, 200.
- *gmax*: 50, 100, 200.
- *Ps*: 0.3, 0.6, 0.97.
- W: 0.1, 0.5, *random*(0.1,0.5).
- C_1, C_2: 1.5, 2.0, *random*(1.5,2.0).

In this way, the total number of combinations was: 729. We used the Success Counting (SCC) measure of performance (defined in Section 2.4.4, page 25), and we performed 30 runs for each combination and for each one of seven different test functions (21870 runs per function, 153090 runs in total): ZDT1, ZDT2, ZDT3, ZDT4, DTLZ2, DTLZ4 and DTLZ6 (previously defined). All the figures corresponding to the obtained results can be consulted in the Appendix B, page 173.

The ANOVA provided the following conclusions:

- *swarmsize*: large values give better results. See Figure B.1.
- *gmax*: as *swarmsize*, large values give better results, although at a lower rate. See Figure B.2.

In fact, from the ANOVA we were able to conclude that it is better to use a large swarm size than a high number of generations. See Figure B.3.

- *Ps*: except for function DTLZ4, large values give better results. See Figure B.4.
- W: except for function DTLZ2, large values give better results. See Figure B.5.

- C_1: this parameter is not important for our MOPSO approach, that is, its value doesn't affect the performance of the algorithm. In Figure B.6, we can see that the use of different values for the parameter C_1 does not affect the performance of the algorithm in a significant way.

- C_2: except for function DTLZ2, large values give better results. See Figure B.7.

Besides these conclusions, from the results obtained by the ANOVA, we could observe some interesting correlations among some parameters. The two most significant were:

- Large values of W not only provide better performance, in general, but also decrease the impact of parameter C_2. See Figure B.8.

- Large values of C_2 not only provide better performance, in general, but also increase the impact of parameter Ps. See Figure B.9.

As we can see, the parameters Ps, W and C_2 are the most important for our approach, since their values affect the performance of the algorithm (that is, their best value in order to obtain a good performance, is problem dependent). In this way, we performed another ANOVA in which the value of the parameters *swarmsize*, *gmax* and C_1 were fixed at: 200, 100 and *random* (1.5,2.0), respectively. In this case, we considered the following levels for the parameters that were found to be the most important:

- Ps: 0.0, 0.2, 0.4, 0.6, 0.8, 0.97, 1.0 (7 levels).

- W: 0.1, 0.2, 0.3, 0,4, 0.5, *random* (6 levels).

- C_2: 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, *random* (7 levels).

In this way, this time we had a total of 294 combinations. We performed 30 runs for each combination and for each one of the seven test functions previously defined (8820 runs per function, 61740 runs in total). As in the case of the previous ANOVA, we obtained very similar conclusions. However, in this case, we were able to obtain the values that gave the best performance of our MOPSO approach:

- *Ps*: 0.2, 0.8, 0.97. See Figure B.10.
- W: 0.2, 0.4, 0.5. See Figure B.11.
- C_2: 1.5, 2.0. See Figure B.12.

These values were used for designing the on-line adaptation mechanism described in the next section.

4.6 On-Line Adaptation

In this section, we describe the on-line adaptation mechanisms proposed for setting the values of the parameters of our MOPSO approach. Given the properties of the concept of ε-dominance, we proposed one adaptation scheme for the ε parameter and another different scheme for the rest of the parameters.

4.6.1 ε Adaptation Mechanism

As we mentioned before, when using our MOPSO approach, for each problem, the value of ε has to be tuned based on the desired number of points in the final Pareto front. In this section we describe a relatively simple mechanism to adapt the value of the parameter ε throughout the run.

The mechanism implemented is based on the formula [58]:

$$\text{PF} \leq \left(\frac{\log K}{\log(1+\varepsilon)} \right)^{m-1},$$

where PF is the number of points desired in the final Pareto front, K is an upper bound for all the objective functions (calculated over the current values) and m is the number of objective functions.

Our approach initializes the value of ε using the previous formula. Later on, the value of ε is adjusted based on the number of points that the current Pareto front contains. At each generation, the number of solutions in the current Pareto front is compared with the desired number of solutions. Then, since large values of ε produce Pareto fronts with few solutions (and viceversa), the value of ε increases or decreases depending on the current number of solutions in the Pareto front. The change in the value of ε is proportional to the difference between the desired and the current number of optimal solutions. For example, if the desired number of

solutions is 10 and there are 12 solutions in the current Pareto front, the value of ε is increased by 20% of its current value.

4.6.2 Adaptation Mechanism for Other Parameters

As we observed in Section 4.5, from the two ANOVAs performed we obtained a set of values for the most important parameters of our MOPSO approach, that give the best performance. Since we have a finite set of possible values for the parameters *Ps*, W and C_2, we considered the problem of the selection of the best value for each parameter as a multi-armed bandit problem.

The multi-armed bandit problem was originally described by Robbins [84]. A multi-armed bandit, also called K-armed bandit, is similar to a traditional slot machine (one-armed bandit) but, in general, has more than one lever. Initially, the gambler has no knowledge about the levers, but through repeated trials, he can focus on the most rewarding levers. There are two versions of the K-armed bandit, the *opaque* in which a unique reward is observed at each round, and the *transparent* in which all rewards are observed. The gambler plays iteratively one lever at each round and observes the associated reward. His objective is to maximize the sum of the collected rewards. The problem of determining the best strategy for the gambler is called the multi-armed bandit problem and many strategies or algorithms have been proposed as solutions to this problem.

In our case, each lever corresponds to one of the possible values for the parameters that we want to adapt. Each time the MOPSO approach selects a specific value (a lever) for a parameter, that value (lever) receives a reward. We define two types of reward:

- **Reward 1**: If the obtained particle is able to enter into the set of leaders, the lever receives a reward of 1. Otherwise, the lever receives a reward of 0.

- **Reward 2**: If the obtained particle is able to enter into the set of leaders, the lever receives a reward of 0.5. Furthermore, if the obtained particle is able to dominate at least one member of the set of leaders, the lever receives an additional reward of 0.5. Otherwise, the lever receives a reward of 0.

All the levers, for the three parameters (*Ps*, W and C_2), start with a total reward of zero. The first generation (zero) of the MOPSO approach is used to obtain initial values for the mean reward for each lever, of each parameter. That is, any specific strategy is applied beginning with generation 1.

We used three different mechanisms to adapt the corresponding values of the studied parameters. The first is proposed by us, and the other two were chosen based on the work presented in [107]:[2]

- **Proportional**. This strategy consists of a random choice according to the probability:

$$p_k = \frac{\mu_k}{\sum_{i=1}^{n} \mu_i}$$

  where μ_i is the estimated mean of the rewards brought by the lever i and n is the total number of levers.

- **The ε-Greedy Strategy**. This is probably the simplest and the most widely used strategy to solve the bandit problem as it was described by Watkins [110]. The ε-greedy consists of choosing a random lever with ε-frequency, and otherwise choosing the lever with the highest estimated mean. ε must be in the open interval (0,1) and its choice is left to the user. The ε value controls the amount of exploration (the probability of executing actions other than the one with the highest estimated mean). In this way, the ε-greedy strategy is sub-optimal because asymptotically, the constant factor ε prevents the strategy from getting arbitrarily close to the optimal lever.

- **The Soft Max Strategy**. This strategy, also called Boltzmann Exploration [61], consists of a random choice according to a Gibbs distribution. The lever k is chosen with probability

$$p_k = \frac{e^{\mu_k/\tau}}{\sum_{i=1}^{n} e^{\mu_i/\tau}}$$

  where μ_i is the estimated mean of the rewards brought by the lever i and $\tau \in \mathbb{R}^+$ is a parameter called the *temperature*. The choice of τ's value is left to the user. The parameter τ has an impact similar to ε. Small values of τ increase the tendency to choose the lever with the best estimated mean.

We tested the proposed mechanisms using the test functions ZDT1, ZDT2, ZDT3, ZDT4, DTLZ2, DLZT4 and DTLZ6 (previously defined). For the ε-greedy and the softmax approach, the allowable values of ε and τ are $\{0.05, 0.10, 0.15\}$, based on the work presented in [107], and given the impact of these parameters

[2] In [107], Vermorel et al. provided the first preliminary empirical evaluation of several multi-armed bandit algorithms.

(briefly discussed in the previous section). We performed 30 runs for each function and each mechanism. The parameters adopted for the MOPSO algorithm were: 200 particles, 100 generations and 100 points in the final Pareto front.[3] The obtained results, using the SCC measure of performance, are shown in Tables B.1 to B.7 (Appendix B, page 173). For each test function, we present the best, median, worst, mean and standard deviation of the SCC measure, for the MOPSO approach without adaptation and the MOPSO approach with adaptation, with the different mechanisms and the two types of reward. The results shown in this section were published in [82].

Function ZDT1. As we can see in Table B.1, all the approaches obtained very good results with both types of reward. However, only the proportional strategy was able to improve the results of the approach without adaptation, by almost 10 particles (on average), using reward 2. It is very interesting to note that in all cases, the approaches with adaptation improved the worst results of the approach without adaptation (except for softmax, whose quality was the same when $\tau = 0.1$). In this way, the standard deviation of the approaches with adaptation was smaller than for the approach without adaptation. In general, the ε-greedy and softmax strategies, had a better performance (on average) using reward 1, being the softmax better than the ε-greedy strategy. Also, we can observe that the ε-greedy strategy lost quality when using reward 2.

Function ZDT2. As in function ZDT1, in this function the proportional approach (using reward 2) obtained the best results having the same quality than the results of the approach without adaptation (see Table B.2). In this case, the softmax strategy had a better performance using reward 2, being better than the ε-greedy strategy (whose best results were obtained using reward 1). As in the previous function, we can observe again that the ε-greedy strategy lost quality when using reward 2.

Function ZDT3. From Table B.3, we can conclude that the proportional approach obtained again the best results using reward 2 and also slightly improved the results obtained by the approach without adaptation. In this case, the ε-greedy

[3] It should be noted that, based on the results obtained from the ANOVA described in the previous section, we were able to improve the performance of our algorithm by changing the confi guration of the values used: in all the previous experiments, we used a swarm of 100 particles for 200 generations. The ANOVA indicated that better results could be obtained by using a swarm of 200 particles for 100 generations.

and the softmax strategies had a best performance using reward 1, being ε-greedy marginally better. Finally, we can observe that the ε-greedy strategy lost quality again when using reward 2.

Function ZDT4. As we can see in Table B.4, in this function all the approaches had a very similar behavior. All the adaptation strategies improved the results obtained by the approach without adaptation or at least reached the same quality. The only exception was the ε-greedy strategy when ε=0.05 and it used reward 2. Also, this same case was the only that didn't improve the worst results obtained by the approach without adaptation. In this case, the best results were from the softmax strategy using reward 1. However, in this case there are no important differences in the performance of the approaches, when compared with respect to the type of reward used.

Function DTLZ2. In this function (see Table B.5) all the adaptation strategies had again very similar behavior. However, only the softmax strategy (using reward 1) was able to improve the results obtained by the approach without adaptation. In this case, all the strategies had better results using reward 1, improving the worst results from the approach without adaptation, in all cases.

Function DTLZ4. As in function ZDT4, in this case all the adaptation strategies improved the results obtained by the approach without adaptation or at least reached the same quality (see Table B.6). Also, all the approaches had their best performance using reward 1, being the best the ε-greedy strategy, in this case. It is very interesting to note that, in this function, all the adaptation strategies significantly improved the best result obtained by the approach without adaptation, specially in the case of the ε-greedy strategy (almost 70 particles, when ε=0.05 and using reward 1). However, since the median and worst values were not improved, the standard deviations were greater in this case.

Function DTLZ6. As we can see in Table B.7, in this function it is again reward 1 the one that provided the best results, for all the adaptation strategies. The best results, in this case, were obtained by the softmax strategy. In this function, the softmax strategy improved the best and median results obtained by the approach without adaptation. However, as in function DTLZ4, since the worst values were not improved, the standard deviations were greater in this case.

Overall discussion. As we observe, the proportional strategy obtained very good

results in three of the seven test functions, using reward 2: ZDT1, ZDT2 and ZDT3. However, this behavior was not consistent. In general, reward 1 provided better results than reward 2, specially in the case of the ε-greedy strategy. The obtained results seem to indicate that the knowledge about the ability of a set of parameters to provide a nondominated particle is enough information to reward it. In general, the softmax strategy had better performance than the proportional and the ε-greedy strategies. In fact, unlike the case of the two other approaches (specially the ε-greedy), the results of the softmax strategy were not significantly affected by the use of different rewards. Also, the softmax strategy was able to improve the performance (on average) of the approach without adaptation in five of the seven test functions, and to maintain the quality of the obtained solutions in one more test function. Finally, for all the test functions, at least one of the three proposed strategies was able to improve the performance (on average) of the approach without adaptation.

4.7 Test Problems Analysis

With the aim of exploring the abilities (or difficulties) of our MOPSO approach for solving different types of test functions, in this section we present a brief analysis of a set of eight different functions, including the seven test functions previously used, which were taken from the specialized literature on evolutionary multi-objective optimization. The test functions considered are: ZDT1, ZDT2, ZDT3, ZDT4, ZDT6, DTLZ2, DTLZ4 and DTLZ6 (defined in Section 2.4.4, page 25).

- **Test Function ZDT1**: This 30-variable problem has a convex and continuous true Pareto front. Figure 4.7 shows the true Pareto front of this function and a set of 30000 randomly generated solutions. As we can see, the density of solutions in the search space, and across the true Pareto front, is uniform.

- **Test Function ZDT2**: This problem has also 30 decision variables, but a nonconvex true Pareto front. This test function is considered the nonconvex counterpart of ZDT1. Figure 4.8 shows the true Pareto front of this problem, and also a set of 30000 randomly generated solutions. As in the case of function ZDT1, we can observe a uniform density of solutions. The difficulty with this problem is that the true Pareto front is nonconvex.

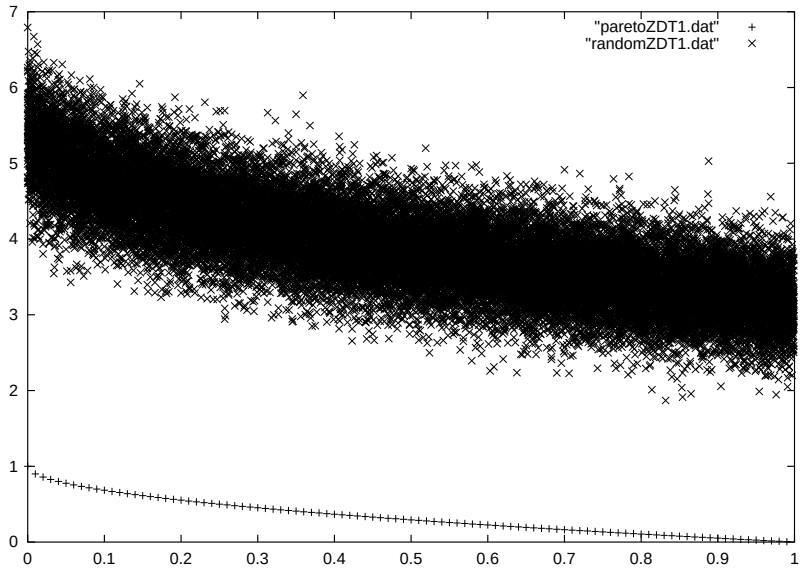


Figure 4.7: True front and 30000 randomly generated solutions, for test function ZDT1.

- **Test Function ZDT3**: This 30-variable problem represents the discreteness feature. The true Pareto front of this test function consists of several disconnected convex parts. Figure 4.9 shows the true Pareto front of this problem, and 30000 randomly generated solutions. As we can see, the density of the solutions is uniform. This problem tests the ability of a MOEA to find all parts of the true Pareto front with a uniform spread.

- **Test Function ZDT4**: This function has 10 decision variables and a convex true Pareto front. The difficulty of this problem is that there exist 21^9 or about $8(10^{11})$ local Pareto optimal solutions in the decision variable space, making a total of 100 distinct local Pareto fronts in the objective space, of which only one is global. Figure 4.10 shows the true Pareto front of this problem and 30000 randomly generated solutions. Function ZDT1 and function ZDT4 have the same true Pareto front. However, if we compare Figure 4.10 and Figure 4.7, we can see that such property is not very clear because, in the case of function ZDT4, the randomly generated solutions are located far away from the true Pareto front, when compared to the case of function ZDT1. This problem tests the ability of a MOEA to deal with

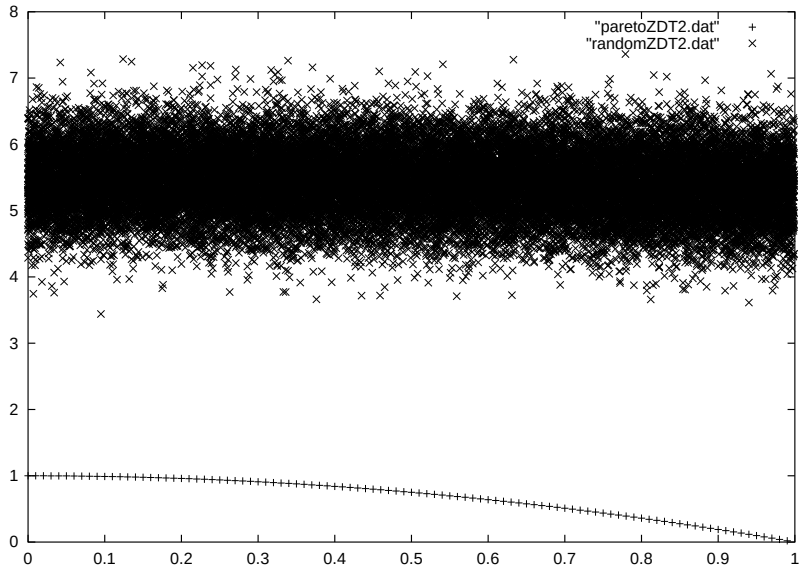


Figure 4.8: True front and 30000 randomly generated solutions, for function ZDT2.

multimodality.

- **Test Function ZDT6**: This 10-variable problem has a nonconvex true Pareto front (the same front of function ZDT2). Figure 4.11 shows 100 uniformly distributed Pareto optimal solutions (in search space $x_1 \in [0,1]$), and 30000 randomly generated solutions. As we can see, the density of solutions in the search space is nonuniform. In fact, the Pareto optimal solutions are nonuniformly distributed along the global Pareto front (the front is biased for solutions for which f_1 is near one). Thus, the nonuniform density of solutions and the nonconvex nature of the Pareto front may cause difficulties for a MOEA to converge to the true Pareto front.

- **Test Function DTLZ2**: This test function has 12 variables and a nonconvex true Pareto front. Figure 4.12 shows the true Pareto front of this problem and 30000 randomly generated solutions. As we can see, the density of solutions is uniform. This problem tests the ability of a MOEA to scale up its performance with a large number of objectives.

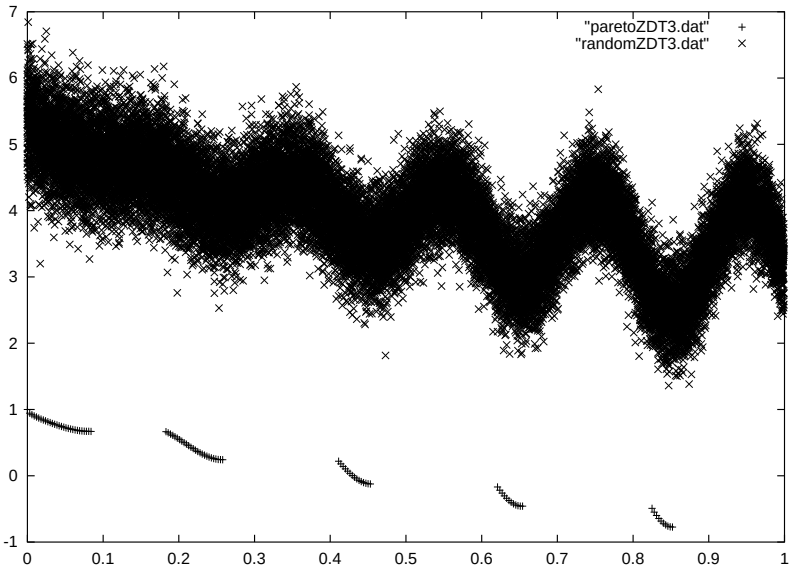


Figure 4.9: True front and 30000 randomly generated solutions, for function ZDT3.

- **Test Function DTLZ4**: As in the case of function DTLZ2, this test function has 12 decision variables and a nonconvex true Pareto front. Figure 4.13 shows the true Pareto front of this problem and 30000 randomly generated solutions. As we can see, the true Pareto front of this function is the same that in Function DTLZ2. However, in this case, the density of solutions is nonuniform. There exists a dense set of solutions near the $f_3 - f_1$ and $f_2 - f_1$ planes. This problem tests the ability of a MOEA to maintain a good distribution of solutions (as in the case of function ZDT6).

- **Test Function DTLZ6**: This 22-variable problem has a true Pareto front that consists of four disconnected regions. Figure 4.14 shows the true Pareto front of this problem and 30000 randomly generated solutions. This problem tests the ability of a MOEA to maintain distributed subpopulations in different Pareto-optimal regions.

As we could see in Section 4.4, our MOPSO approach provided results of good quality in almost all the test functions used. However, the quality of the results obtained for the cases of functions DTLZ2 and DTLZ4 was not good enough. In

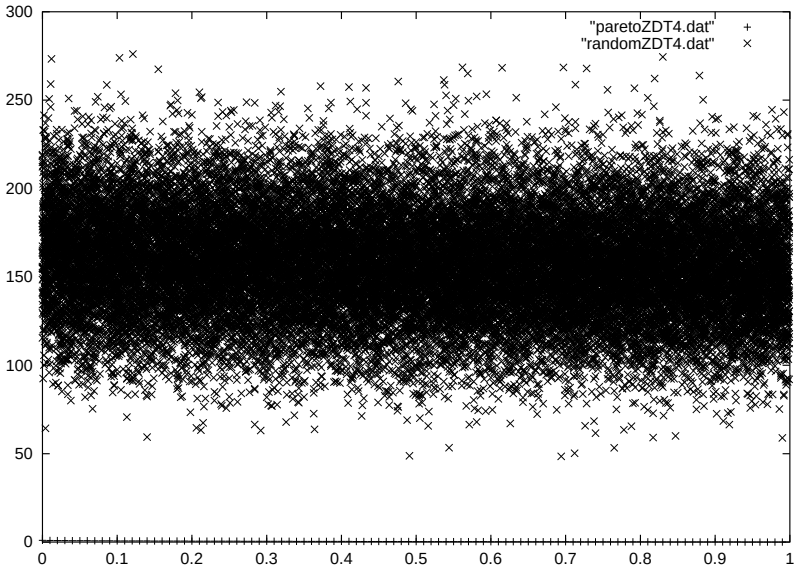


Figure 4.10: True front and 30000 randomly generated solutions, for function ZDT4.

fact, the worst results provided by our algorithm were those obtained for function DTLZ4.

Functions DTLZ2 and DTLZ4 have both three objectives and 12 variables. Thus, the first feature that may cause these functions being hard to optimize is the number of objectives. Nevertheless, function DTLZ6 has also three objectives and our approach had a good results on that test function. This behavior might be due to the fact that function DTLZ6 has the same (design) structure of functions ZDT. Thus, we argue that the combination of the number of objectives and the nonlinearity of functions DTLZ2 and DTLZ4 is the reason for these functions being “difficult” test functions.

The case of function DTLZ4 is special. As we mentioned before, this function was the most difficult to optimize, for our MOPSO approach. We think that this behavior of our algorithm is caused by the nonuniform density of the solutions in the search space defined by function DTLZ4, as we can see in Figure 4.13.

As it was described before, function ZDT6 has a nonuniform density of solutions in the search space. For this reason, although function ZDT6 was not used in the validation of our MOPSO approach, we decided to test our algorithm on

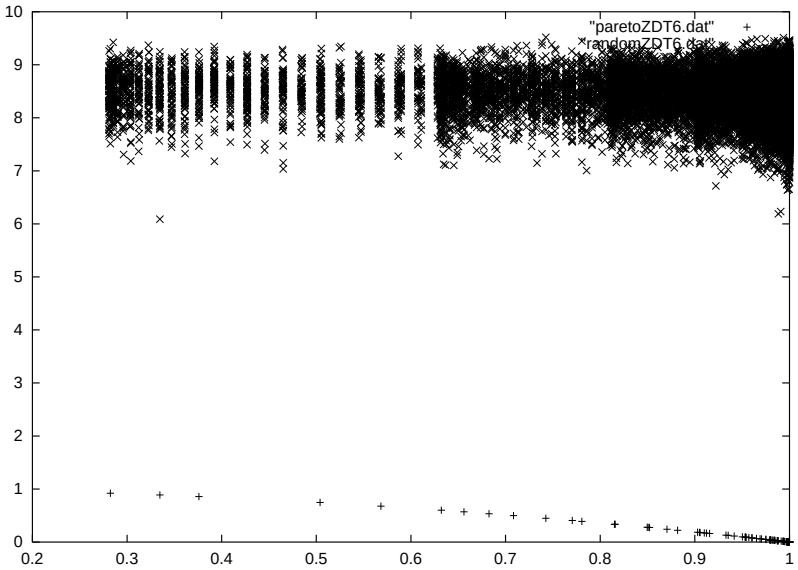


Figure 4.11: True front and 30000 randomly generated solutions, for function ZDT6.

this function, as part of this analysis. Figure 4.15 shows the true Pareto front of function ZDT6 and the union of 30 Pareto fronts generated by our approach, after 20000 evaluations of the objective functions. As we can see, this function is hard to optimize, for our MOPSO approach.

Therefore, it seems that our approach has some difficulties optimizing functions with a nonuniform density of solutions in the search space. Test functions with this feature, test the ability of a MOEA to find a good distribution of solutions, despite the natural non-uniform density of solutions in the search space. That is, in these cases, a MOEA should be able to maintain diversity.

These conclusions seem to agree with the behavior of our algorithm, given the following discussion. As we mentioned in Section 4.3, the final version of our approach uses a leader selection technique obtained from the combination of two different techniques: technique I and technique II. Technique I is based only on dominance relations and technique II was designed to favor diversity into the swarm (based on the crowding factor of each leader). The first version of our MOPSO approach only adopted leader selection technique II. Figure 4.16 shows the nondominated solutions obtained from the union of 20 Pareto front

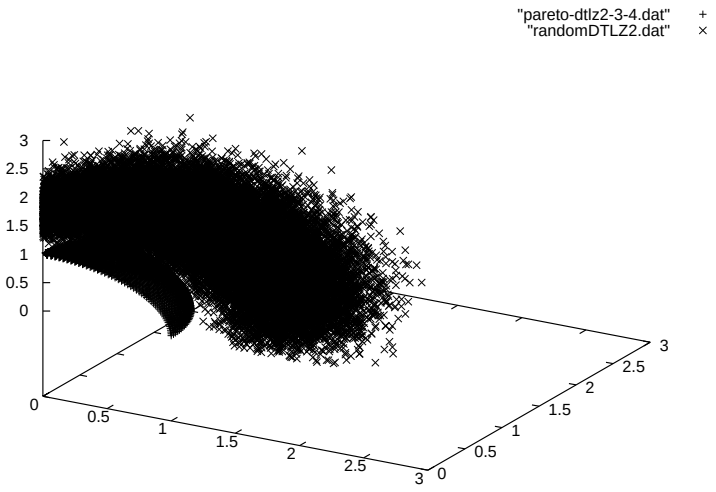


Figure 4.12: True front and 30000 randomly generated solutions, for function DTLZ2.

generated by the first version of our MOPSO approach, after 20000 evaluations of the objective functions. As we can see, the previous version of our approach had better results on function DTLZ4 than the final version. More precisely, while the final version of our approach obtained an average of 11 Pareto optimal solutions for function DTLZ4, the previous version was able to obtain an average of 43 optimal solutions (these results can be seen in [95]).

This behavior also agrees with the results obtained from the ANOVA previously described in Section 4.5. The performed ANOVA indicated that, for function DTLZ4, the value of the *Ps* parameter should be small in order to obtain better results. Parameter *Ps* determines the probability with which the leader is selected using technique I, otherwise, technique II is used. That is, parameter *Ps* indicates the probability with which leaders are selected based only on dominance relations, otherwise, leaders are selected based on diversity measures. Thus, small values of parameter *Ps* favor diversity into the swarm. In this way, small values of parameter *Ps* may improve the performance of our MOPSO approach in functions with nonuniform density on the search space.

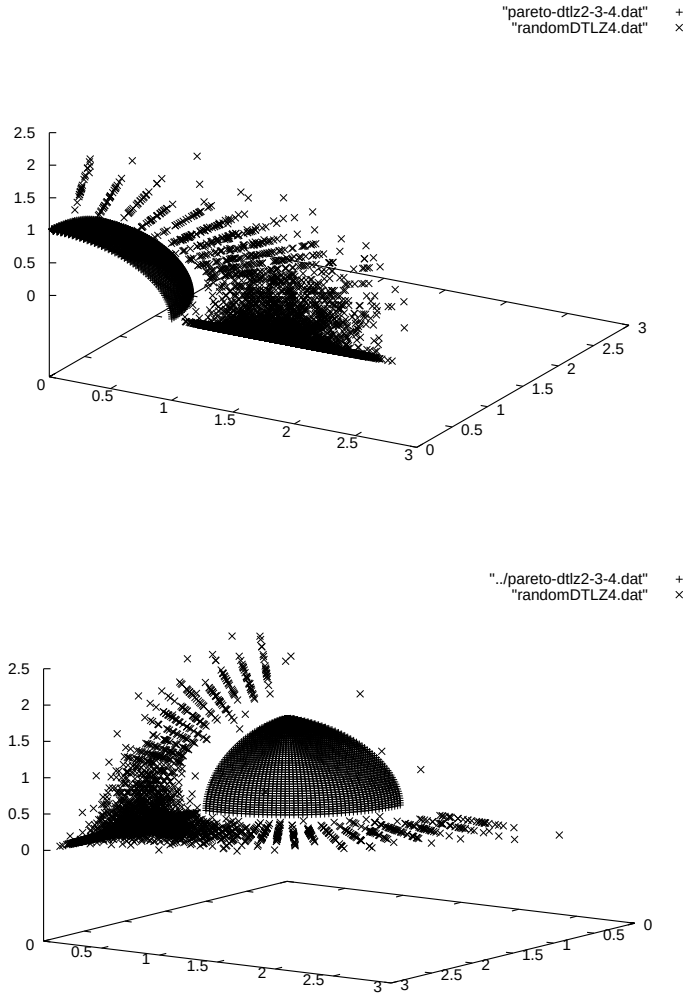


Figure 4.13: True front and 30000 randomly generated solutions, for function DTLZ4.

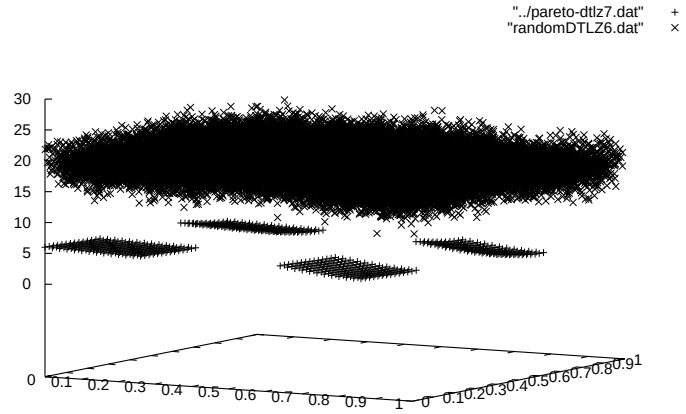


Figure 4.14: True front and 30000 randomly generated solutions, for function DTLZ6.

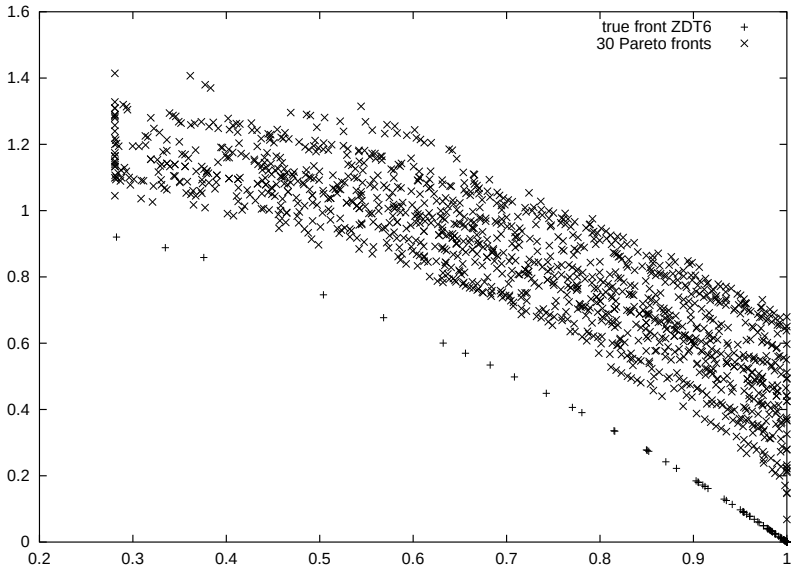


Figure 4.15: True front ($+$) and the union of 30 Pareto fronts ($\times$) generated by the final version of our approach, for function ZDT6.

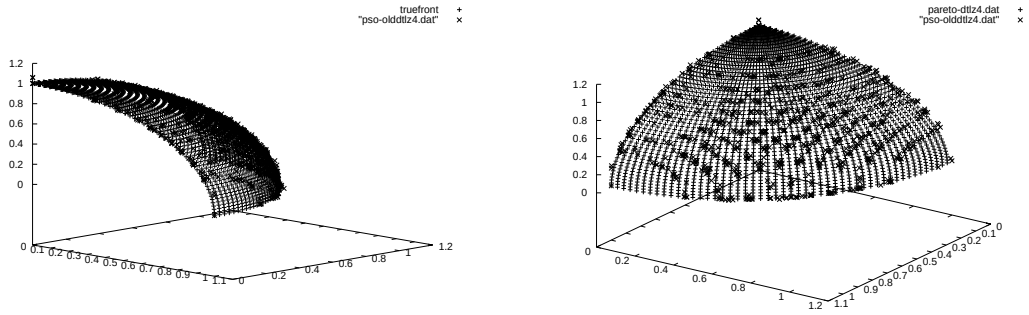


Figure 4.16: True front and the nondominated solutions obtained from the union of 20 Pareto fronts generated by the first version of our approach, for function DTLZ4.

As we can see in the definition of functions ZDT6 and DTLZ4, nonuniform density in the search space is caused by the high nonlinearity of the objective functions: $\sin^6$ (f_1) in ZDT6, and $\alpha = 100$ (f_1, f_2 and f_3) in DTLZ4. We performed an experiment to investigate the variation in the values of the objective functions of problems ZDT6 and DTLZ4, comparing them to the corresponding variations in problems ZDT2 and DTLZ2, respectively. For each test function, we evaluated $d(\mathbf{f}(\mathbf{x}),\mathbf{f}(\mathbf{x}+\delta))$, where d is the Euclidean distance and $\varepsilon \in \{0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001\}$. For each different value of δ, we evaluated the value of d for 100 different $\mathbf{x}$ vectors randomly generated. Tables 4.1 and 4.2 show the standard deviations of the values obtained, for each value of δ. As we can see, the standard deviations for functions ZDT6 and DTLZ4 are much higher than the corresponding values of functions ZDT2 and DTLZ2, respectively. Thus, these results indicate the variability (nonlinearity) of the different test functions.

In this way, we can conclude that, given the fixed value of parameter *Ps* (0.97), the final version of our MOPSO approach has some difficulties optimizing functions with high nonlinearity. This is due to the fact that high nonlinearity in the objective functions defines search spaces with nonuniform density. In those cases, diversity is the key to obtain good approximations of the true Pareto front. Therefore, in functions with such characteristics, a leader selection technique based on diversity measures is needed. Thus, based on the previous analysis, we finally conclude that when using our MOPSO approach to optimize high nonlinear func-

Test Function ZDT2						
ε	0.1	0.01	0.001	0.0001	0.00001	0.000001
std. dev.	0.0022059	0.0002526	0.0000256	0.0000025	0.0000004	0.0000000
Test Function ZDT6						
ε	0.1	0.01	0.001	0.0001	0.00001	0.000001
std. dev.	0.0862285	0.0403831	0.00429982	0.0004316	0.0000432	0.0000043

Table 4.1: Standard deviations in the objective values of functions ZDT2 and ZDT6.

Test Function DTLZ2						
ε	0.1	0.01	0.001	0.0001	0.00001	0.000001
std. dev.	0.0897617	0.0082031	0.0008242	0.00008252	0.0000082	0.0000008
Test Function DTLZ4						
ε	0.1	0.01	0.001	0.0001	0.00001	0.000001
std. dev.	1.0311648	0.2692921	0.0184366	0.00176306	0.0001755	0.0000175

Table 4.2: Standard deviations in the objective values of functions DTLZ2 and DTLZ4.

tions, the user may try using small values for the parameter P_s in order to obtain better approximations of the true Pareto front.

Finally, as we mentioned in Section 4.4, it is important to note that these conclusions are restricted to the set of functions used, since by the No Free Lunch Theorem [112], general conclusions about the behavior of our algorithm can not be possibly drawn.

4.8 Constrained Multi-Objective Optimization

In this section, we provide some results of a set of experiments performed to test the ability of our MOPSO approach to deal with constrained search spaces. Our MOPSO approach incorporates constraints by modifying the dominance relation:

- When two particles are feasible, the previous defined Pareto dominance is checked.
- When one particle is feasible and the other one is not, the feasible dominates the infeasible.
- When both particles are infeasible, the one with the lowest sum of violations dominates the other.

Table 4.3 provides the list of functions used, with the corresponding number of variables, objectives and constraints. Also, Table 4.3 indicates the number of evaluations performed for each one of the test functions. Since function Osyczka is considered the hardest of the seven functions used, it was the only on which our algorithm performed 20000 evaluations. Functions Kita and Tanaka were defined in Section 2.4.3 (page 19), and the definitions of the rest of the functions can be found in [21].

Table 4.4 shows the results obtained by our algorithm using the SCC and IGD measures of performance (previously defined). Also, Figure 4.17 shows the Pareto front corresponding to the median result with respect to the IGD measure, for each function. As we can see in Table 4.4, function Osyczka was the only in which our algorithm was not able to obtain optimal solutions. Also, from the rest of the functions, Jimenez was the hardest to optimize. However, as we can see in Figure 4.17, our algorithm was able to obtain very good approximations of the true Pareto front of all functions, including Jimenez and excepting Osyczca. In the case of function Osyczca, our algorithm obtained a partial approximation of

Function	variables	objectives	constraints	evaluations
Srinivas	2	2	2	5000
Belegundu	2	2	2	5000
Kita*	2	2	3	5000
Jimenez*	2	2	4	5000
Binh	2	2	2	5000
Tanaka	2	2	2	5000
Osyczka	6	2	6	20000

Table 4.3: Constrained test functions used to validate our MOPSO approach. Functions marked with an asterisk (*) are maximization problems.

		Srinivas	Belegundu	Kita	Jimenez	Binh	Tanaka	Osyczka
SCC	best	95	100	92	46	93	98	0
	median	86	98	85	33	86	78	0
	worst	75	92	78	17	79	60	0
	mean	**85**	**98**	**85**	**32**	**86**	**77**	**0**
	st. dev.	5.1	1.7	3.5	6.2	3.5	9.3	0
IGD	best	0.0305	0.0019	0.0023	0.0142	0.1746	0.0009	0.1138
	median	0.0333	0.0019	0.0026	0.0149	0.1843	0.0013	0.2514
	worst	0.0395	0.0019	0.0032	0.0168	0.1883	0.0018	0.3967
	mean	**0.0336**	**0.0019**	**0.0026**	**0.0150**	**0.1837**	**0.0013**	**0.2366**
	st. dev.	0.0020	0.0000	0.0002	0.0006	0.0038	0.0003	0.0635

Table 4.4: Results obtained by our approach, in constrained functions.

the true Pareto front. However, being function Osyczka a very hard function, we can consider this as a good result.

4.9 Coevolutionary Multi-Objective Particle Swarm Optimization

As we mentioned in Chapter 3, the MOPSO approach described and studied in the previous sections was designed with the aim of incorporating it as a search engine into the coevolutionary scheme described in Chapter 3.

In this section, we provide some results of the coevolutionary scheme provided

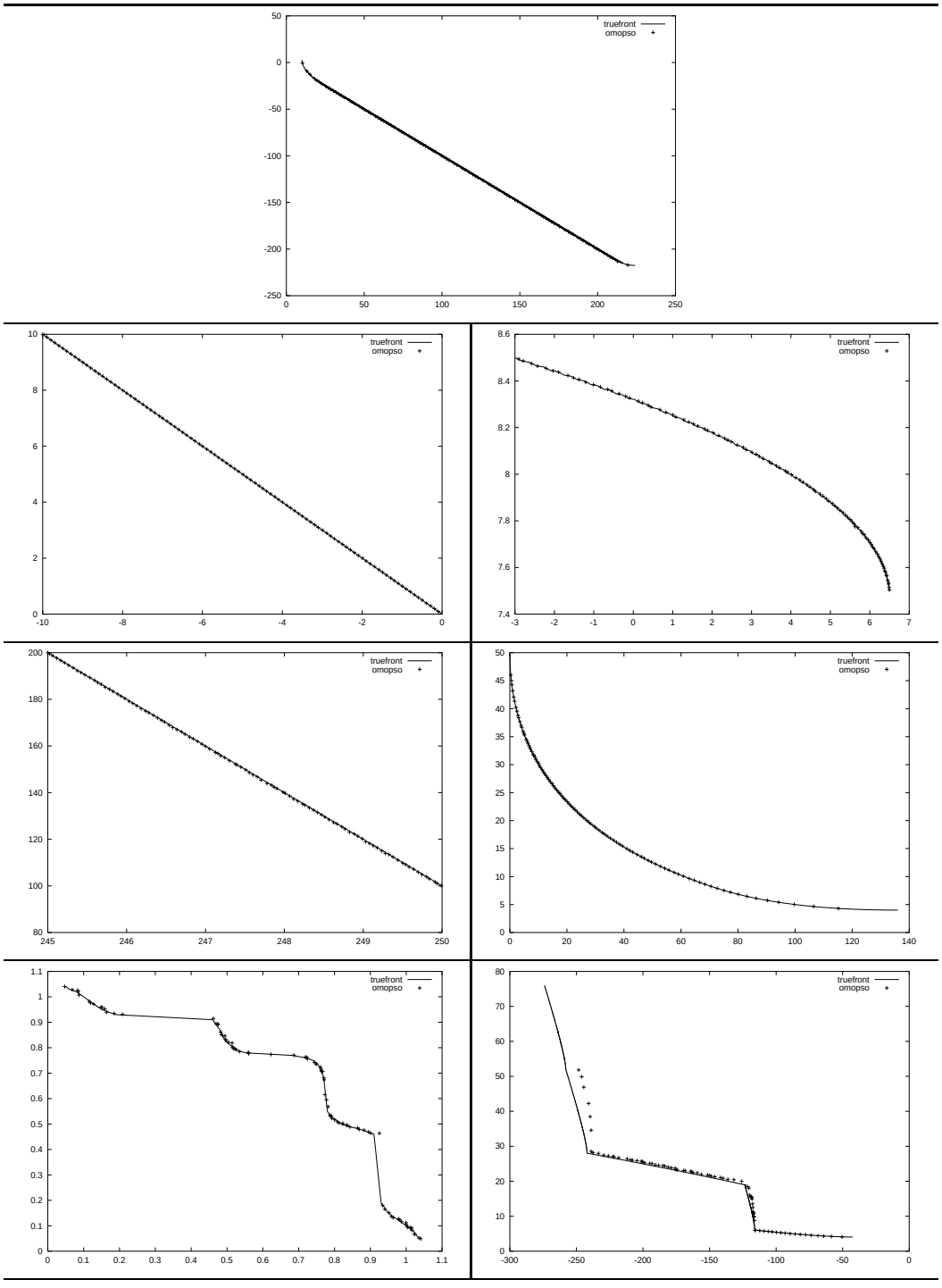


Figure 4.17: Results obtained by our approach, in constrained functions.

		ZDT1	ZDT2	ZDT3
SCC	best	77	73	51
	median	43	15	11
	worst	2	0	2
	average	33	20	17
	std. dev.	24.5	23.5	12.5

Table 4.5: Results obtained by our approach in high-dimensional functions.

in Chapter 3 with the MOPSO approach previously defined, as a search engine. We performed 20 runs and a total of 20,000 evaluations (on average), for functions ZDT1, ZDT2 and ZDT3 (previously defined).

Table 4.5 shows the results obtained by our algorithm, with respect to the SCC measure. Also, Figure 4.18 shows the solutions obtained by our coevolutionary approach, for the three functions. The plots on the left show the union of the 20 Pareto fronts obtained. The plots on the right show the nondominated solutions produced from the union of the 20 Pareto fronts obtained.

As we can see in Table 4.5, the results of the SCC measure indicate that our algorithm was able to obtain some optimal solutions, for the three functions considered. However, although the results obtained by this new version of our coevolutionary approach are better than those of the previous one, we can see that the quality of these new results is not better than that of the corresponding results of the MOPSO approach itself. On the other hand, as we can see in Figure 4.19, our coevolutionary approach still has problems covering the whole Pareto front. In fact, in function ZDT2, our algorithm collapses on just one point in several runs.

From the obtained results, we conclude that the proposed coevolutionary scheme is not able to improve the results of the MOPSO approach itself, neither to obtain at least the same results while performing the same number of function evaluations. In this way, it seems that the coevolutionary scheme proposed is not able to reduce computational cost, as it was expected, for the test functions used.

4.10 Conclusions

In this chapter, we described a new MOPSO algorithm, designed with the aim of incorporating it, as a search engine, into the coevolutionary approach described in Chapter 3. The MOPSO approach was tested and studied using several differ-

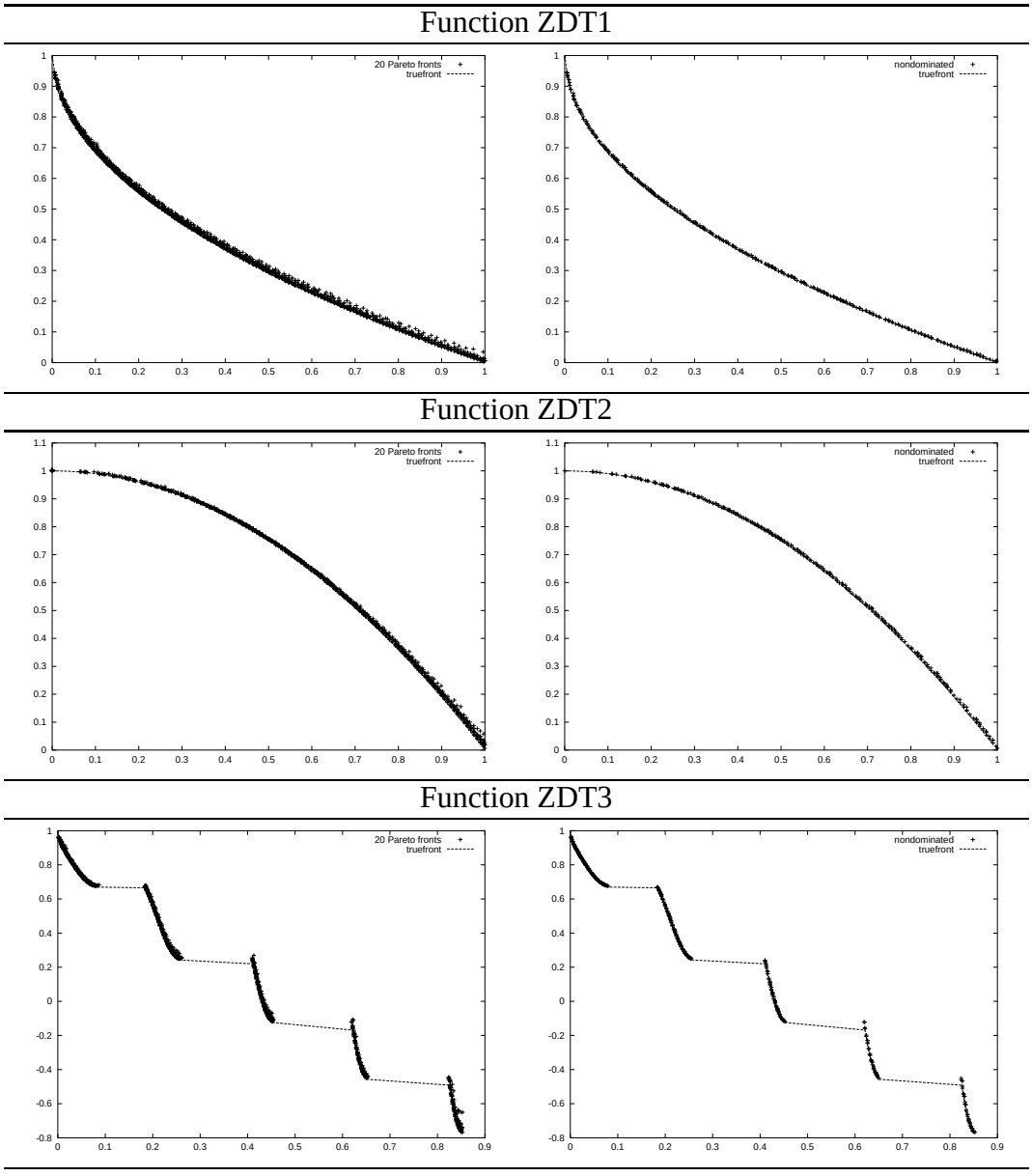
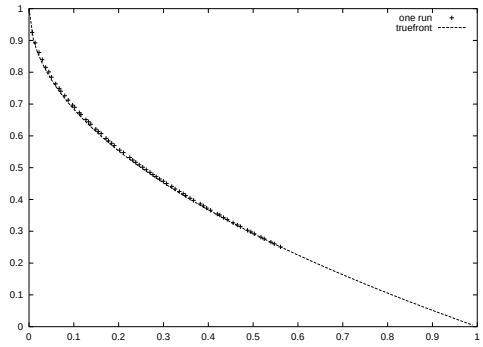
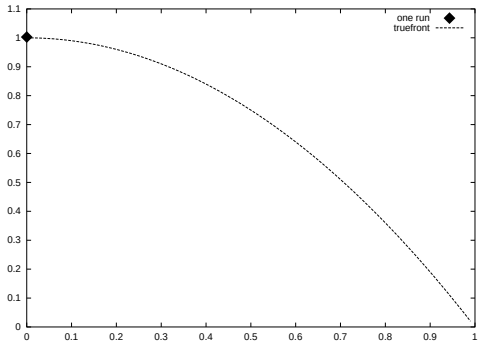


Figure 4.18: Solutions obtained by our coevolutionary approach, for functions ZDT1, ZDT2 and ZDT3. The plots on the left show the union of the 20 Pareto fronts obtained. The plots on the right show the nondominated solutions produced from the union of the 20 Pareto fronts obtained.

Function ZDT1



Function ZDT2



Function ZDT3

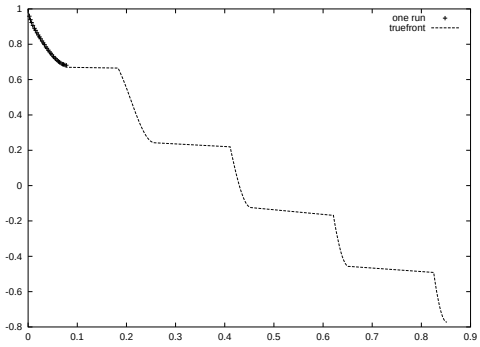


Figure 4.19: Example of one run in which our coevolutionary approach was not able to obtain the whole Pareto front, for functions ZDT1, ZDT2 and ZDT3.

ent test functions taken from the multi-objective optimization literature, and the obtained results were very competitive.

Finally, we proceeded to incorporate the MOPSO algorithm into our coevolutionary approach. That is, we replaced the MOGA algorithm with our new PSO-based approach. Unfortunately, we didn't obtain the expected results. The proposed coevolutionary approach was expected to obtain the same results as those produced by the search engine alone (in terms of the measures of performance used), while performing a smaller number of function evaluations. However, as we could see, the obtained results indicate that the coevolutionary approach was not able to equal the results of the MOPSO approach itself, while performing the same number of function evaluations.

We concluded that the previously proposed coevolutionary approach had some disadvantages. When testing the approach with low dimensional functions the results were reasonably good. However, when using high dimensional functions, the coevolutionary approach degraded the quality of the results obtained by the PSO-based approach alone. In this way, we could see that our coevolutionary approach had problems with high dimensional decision spaces. We argue that this problem is due to the fact that the usefulness of the partition scheme used by our coevolutionary approach depends on the test function adopted. That is, to perform a partition of the search space is, in general, not useful. In particular, the proposed coevolutionary scheme does not seem to be useful for the set of test functions adopted, which consists of very specific continuous real-valued functions (that is precisely the type of problem in which we are interested). However, it remains open the posibility of testing the usefulness of such scheme when using it for solving other types of problems.

For these reasons, we decided to study alternative mechanisms for reducing computational cost. The following chapter describes our work done on this topic.

Chapter 5

Fitness Inheritance in Multi-Objective Particle Swarm Optimization

While studying different mechanisms to reduce computational cost, we found a biological concept previously proposed with that purpose and that had not been applied to Particle Swarm Optimization. In this chapter, we describe our proposal to incorporate the concept of fitness inheritance into the PSO-based approach described in the previous chapter, in order to reduce the number of function evaluations performed.

5.1 Fitness Inheritance

Fitness inheritance is an enhancement technique that has been proposed to improve the performance of EAs [99]. In fitness inheritance, the fitness value of an offspring is obtained from the fitness values of its parents. In this way, we do not need to evaluate every individual at each generation, and therefore, the computational cost is reduced.

The use of fitness inheritance to improve the performance of GAs was originally proposed by Smith et al. [99]. The authors proposed two possible ways of inheriting fitness: the first consists of taking the average fitnesses of the two parents and the other consists of taking a weighted (proportional) average of the fitnesses of the two parents. The second approach is related to how similar the offspring is with respect to its parents (this is done using a similarity measure).

They applied inheritance to a very simple problem (the OneMax problem) [99] and found that the weighted average fitness resulted in a better performance and indicated that fitness inheritance was a viable alternative to reduce the computational cost of a genetic algorithm.

Sastry et al. [91] provide some theoretical foundations for fitness inheritance. They investigated convergence times, population sizing and the optimal proportion of inheritance for the OneMax problem. Chen et al. [12] investigate fitness inheritance as a way to speed up multi-objective GAs and EAs. They extended the analytical model proposed by Sastry et al. for multi-objective problems. Convergence and population-sizing models are derived and compared with respect to experimental results. The authors concluded that the number of function evaluations can be reduced with the use of fitness inheritance.

5.2 Previous Work

There are very few references in the literature in which fitness inheritance is used to reduce computational cost. We will review the main ones in this section:

- Ducheyne et al. [29] tested the performance of average and weighted average fitness inheritance on a well-known test suite of multi-objective optimization problems [116], using a binary GA. They concluded that the fitness inheritance efficiency enhancement techniques can be used in order to reduce the number of fitness evaluations provided that the Pareto front is convex and continuous. They also concluded that if the Pareto surface is not convex or if it is discontinuous, the fitness inheritance strategies fail to reach the true Pareto front.

- Salami et al. [88] proposed a “Fast Evolutionary Algorithm” in which a fitness and associated reliability value are assigned to each new individual that is only evaluated using the true fitness function if the reliability value is below a threshold. Also, they incorporated random evaluation and error compensation strategies. The authors obtained an average reduction of 40% in the number of evaluations while obtaining solutions of similar quality. In the same work, they presented an application of fitness inheritance to image compression obtaining reductions between 35% and 42% of the number of evaluations.

- Bui et al. [10] compared the performance of anti-noise methods, particularly probabilistic and re-sampling methods, using NSGA-II [27]. They applied the concept of fitness inheritance to both types of methods in order to reduce calculation time. The authors obtained a substantial amount of savings in computational time (reaching 30% in the best case), without deteriorating the performance.

As we could see, two of the previously described approaches apply fitness inheritance to solve multi-objective problems, using genetic algorithms [29, 10]. However, it is important to mention that, in the case of the work developed by Bui et al. [10], the motivation for using inheritance was not the high cost of the objective function itself (as it is in our case). In [10], since the use of anti-noise methods increases the number of function evaluations in a considerable way, the authors propose the use of fitness inheritance to overcome such disadvantage. For this reason, we can see fitness inheritance, in this case, as a way of improving the applicability of the anti-noise methods.

On the other hand, in the proposal of Ducheyne et. al [29], the motivation for using fitness inheritance is actually the high cost of the objective function itself. However, their approach doesn't work when the true Pareto front of the problem is not convex or discontinuous.

In the next sections, we will describe several fitness inheritance techniques especially designed for multi-objective optimization problems, and whose applicability is not affected by the topology of the true Pareto front. Also, the proposed tehcniques represent the first attempt to incorporate the use of fitness inheritance into a PSO-based optimization algorithm.

5.3 First Proposed Technique

Since PSO has no recombination operator, we adopted as "parents" of a particle the previous position of the particle and its leader. We performed experiments using the weighted average inheritance over real numbers encoding [99]:

Given the previous position of a particle x_{old}, its assigned leader x_{ld} and the new position x_{new}, we proceed to calculate the distance from x_{new} to its "parents" (as defined before):

$$d_1 = d(x_{\text{new}}, x_{\text{old}}), \quad d_2 = d(x_{\text{new}}, x_{\text{ld}}),$$

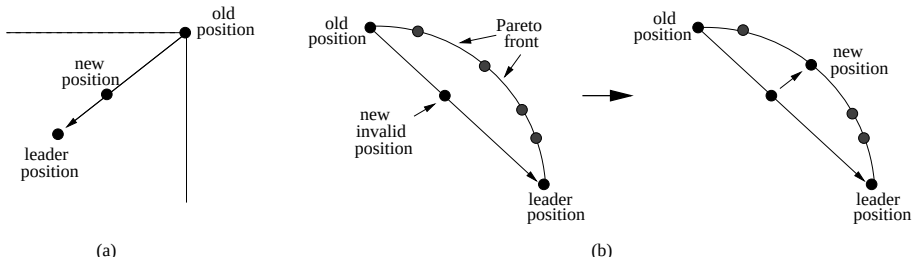


Figure 5.1: The two possible cases of fitness inheritance: (a) when the leader dominates the particle and (b) when the leader does not dominate the particle.

where d is an Euclidean distance in the decision variable space. Later, we calculate how near is the new position from these two positions (x_{old} and x_{ld}) and proceed to inherit the corresponding objective function values:

$$r = \frac{d_1}{d_1 + d_2}$$

$$f_i(x_{\text{new}}) = r\, f_i(x_{\text{ld}}) + (1 - r)\, f_i(x_{\text{old}}), \quad i = 1, ..., n$$

where f_i is the value of the objective function i and n is the number of objective functions. As we can see, the new objective values are linear combinations of the previous values of the particle and the values of the leader.

Ducheyne [29] noted that the previous form of fitness inheritance has problems when the Pareto front is not convex. We came across this same limitation and after some analysis, we concluded that the problem arises in our MOPSO only when the leader chosen does not dominate the current particle. So, we changed the fitness inheritance mechanism when the above situation arises. In this case, we use the values obtained using the original scheme to locate the closest leader to that position. Then, the objective function values of such leader are assigned to the new position. We can see both types of inheritance in Figure 5.1.

In Figure 5.2, the symbol ($\Rightarrow$) indicates the line in which the concept of fitness inheritance is incorporated. The *inheritance proportion*, p_i, is the proportion of individuals in the population whose fitness is inherited. It is very important to note that a particle that has inherited its objective values **can not** enter into the final Pareto front.

```
Begin
   Initialize swarm. Initialize leaders.
   Send leaders to ε-archive
   crowding(leaders), g ← 0
   While g < gmax
      For each particle
         Select leader. Flight. Mutation.
⇒        If(p_i) Inherit Else Evaluation.
         Update pbest.
      EndFor
      Update leaders, Send leaders to ε-archive
      crowding(leaders), g ← g + 1
   EndWhile
   Report results in ε-archive
End
```

Figure 5.2: Pseudocode of our algorithm. The symbol (⇒) indicates the line in which the concept of fitness inheritance is incorporated.

5.3.1 Discussion of Results

The results discussed in this section were published in [97]. We tested our approach using four test functions: ZDT1, ZDT2, ZDT3 and ZDT4 (defined in Section 2.4.3, page 19). We performed experiments with different values of *inheritance proportion* p_i. We experimented with: p_i= 0.1, 0.2, 0.3, 0.4. Note that this proportion of individuals indicates also the percentage by which the number of evaluations is reduced (e.g., $p_i = 0.1$ means that 10% less evaluations are performed). Also, we compared our results against another PSO-based multi-objective approach representative of the state-of-the-art: the Sigma-MOPSO [68]. The approaches will be identified with the following labels: sMOPSO refers to [68], and oMOPSO is our MOPSO [98, 97].

We performed 20 runs for each function and each approach. The parameters of each approach were set such that they all performed 20000 objective function evaluations. In this way, the approach with fitness inheritance performed approximately 18000 evaluations when p_i= 0.1, 16000 when p_i= 0.2, 14000 when p_i= 0.3 and 12000 when p_i= 0.4. All the algorithms were set such that they provided Pareto fronts with 100 points. For our comparative study we used three measures

of performance: Success Counting (SCC), Inverted Generational Distance (IGD) and Two Set Coverage (SC). These measures are defined in Section 2.4.4, page 25. All the tables and figures corresponding to the obtained results can be consulted in Appendix C, page 191.

Tables C.1 and C.2 summarize the results obtained with respect to the unary measures and Table C.3 summarizes the results obtained for the binary measure. The Pareto fronts shown in Figures C.1 and C.2 correspond to the nondominated vectors obtained from the union of the 20 Pareto fronts produced by each approach. It should be noted that the Pareto fronts shown were also used to apply the binary measure of performance.

Tables C.1 and C.2 show that, with respect to the SCC measure, the use of fitness inheritance decreases the quality of the results as we increase p_i. However, with respect to the IGD measure all the approaches with fitness inheritance have almost the same median and mean values that the approach without inheritance, except in function ZDT3. In this way, we can conclude that the approaches with fitness inheritance obtained as good approximations to the corresponding true Pareto front, as the approach without inheritance. In fact, it is very interesting to note that, in some cases, when the value of p_i is low, the quality of the results with respect to the SCC measure is better than the algorithm without fitness inheritance. This is the case of functions ZDT1 (p_i = 0.1), ZDT3 (p_i = 0.1) and ZDT4 (p_i = 0.1,0.2). This effect in the results may be a product of the diversity that is introduced by a particle that has inherited the objective values of a leader. On the other hand, it is always greater the number of function evaluations saved, than the loss of quality in the obtained results (on average, with respect to the SCC measure). For example, in functions ZDT1, ZDT2 and ZDT3, the worst results were obtained by the approach with p_i = 0.4, reducing the quality of the results in a 14.1%, 13.5% and 13.2%, respectively. In function ZDT4, the worst results were obtained by the approach with p_i = 0.3 reducing the quality of the results in a 25%. In fact, even in the worst case with a saving of 40% of evaluations, the results of the approaches with fitness inheritance are better than the results of the another PSO-based approach (sMOPSO).

Since, in general, the performance with respect to the IGD of all the approaches with inheritance in very similar, we choose the approach with the worst results with respect to the SCC measure to be represented by means of its corresponding Pareto front in Figures C.1 and C.2. Thus, in functions ZDT1, ZDT2 and ZDT3, we show the Pareto front corresponding to the approach with p_i = 0.4 and, in the case of function ZDT4, the Pareto front corresponding to the approach with p_i = 0.3. We can see in Figures C.1 and C.2 that the approaches with in-

heritance do not lose the Pareto front even in cases where the true Pareto front is not convex or discontinuous. As we described in Section 5.3, we implemented a simple mechanism (shown in Figure 5.1, page 104) in order to avoid producing invalid particles in the cases of Pareto fronts which are non-convex or discontinuous, this is the main reason why our algorithm does not have problems with these types of test functions.

From the results shown in Table C.3, and following the definitions provided in Section 4.4 (page 69), we can conclude that our algorithm without inheritance (oMOPSO) is *relatively better* than almost all the approaches with fitness inheritance in all functions, except for the case when $p_i = 0.1$. The approach with $p_i = 0.1$ is *relatively better* than the approach without inheritance in three of the four functions: ZDT1, ZDT2 and ZDT3. This conclusion agrees with the results obtained with the unary measures. The only function in which the algorithm without inheritance is *relatively better* than all the approaches with fitness inheritance is ZDT4. These results seem to indicate that the fitness inheritance approach is more useful when the decision space of the problem has a high dimension, since function ZDT4 is the one with the lowest number of variables (function ZDT4 has only 10 variables while the rest have 30 variables).

From our results, we concluded that the efficiency enhancement technique proposed reduces the computational cost without decreasing the quality of the results in a significant way. Also, the fitness inheritance technique used in our approach is able to generate non-convex and discontinuous Pareto fronts. On the other hand, obtained results seem to indicate that the proposed enhancement technique is more useful, that is, the quality of the results is less affected when the decision space is high-dimensional.

Given the obtained results, we decided to explore alternative ways to incorporate fitness inheritance into our PSO-based approach. Such work is described in the following section.

5.4 Study of Different Techniques

Since the results obtained with the first proposed fitness inheritance technique were promising, we decided to study other ways of incorporating such concept into our PSO-based approach. Also, we decided to apply the concept of fitness approximation in order to reduce computational cost. In this way, we proposed a total of nineteen techniques: fifteen fitness inheritance techniques and four approximation techniques. In this section we describe these techniques and the re-

sults obtained.

5.4.1 Fitness Approximation

Another promising possibility when an evaluation is very time consuming or expensive is not to evaluate every individual, but just estimate the quality of some of the individuals based on an approximate model of the fitness landscape.

By using approximation techniques [53], it is possible to estimate the fitness of an individual using the previously calculated fitness of its neighbors. There are many possible approximation models. In the simplest case, the fitness of a new individual is derived from its parents' fitnesses (fitness inheritance). However, there are some other methods like polynomials, the kriging model [9], neural networks [53] and interpolation and regression [8]. Reported experiments [8] show that using fitness estimation, it is possible to either reach a better fitness level in a certain given time, or to reach a desired fitness level much faster. We adopt very simple approximation techniques, based only on the objective values of the closest neighbors.

As in the case of the technique described in the previous section, we can see in Figure 5.2 (page 105) that the symbol ($\Rightarrow$) indicates the line in which the concept of fitness inheritance (or approximation) is incorporated into our MOPSO approach.

5.4.2 Proposed Techniques

Fitness Inheritance

Linear Combination Based on Distances (**LCBD**)

We propose to calculate the new position in the objective space of a particle by means of a linear combination of the positions of the particles that were considered to calculate the new position in the search space.

Given the previous position of a particle x_{old}, its personal best x_{pbest}, the position of its assigned leader x_{ld} and the new position x_{new}, we proceed to calculate the distance from x_{new} to its "parents" (as defined before): $d_1 = d(x_{new}, x_{old}), d_2 = d(x_{new}, x_{pbest}), d_3 = d(x_{new}, x_{ld})$, where d is an Euclidean distance. We propose variants of the same idea, based on the individuals that can be considered. We consider the position of the leader as the most important. Thus, the leader will be always considered:

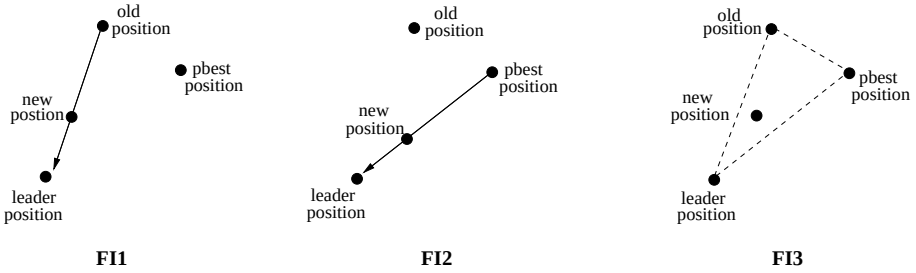


Figure 5.3: Illustration of techniques FI1, FI2 and FI3.

FI1 Previous position and position of the leader: $r = \frac{d_1}{d_1+d_2}$,

$$f_i(x_{new}) = r\, f_i(x_{ld}) + (1-r)\, f_i(x_{old}), i = 1,...,n.$$

FI2 *pbest* position and position of the leader. $r = \frac{d_2}{d_2+d_3}$,

$$f_i(x_{new}) = r\, f_i(x_{ld}) + (1-r)\, f_i(x_{pbest}), i = 1,...,n.$$

FI3 Previous position, *pbest* position and position of the leader.
$r_1 = \frac{d_1}{d_1+d_2+d_3}$, $r_2 = \frac{d_2}{d_1+d_2+d_3}$, $r_3 = \frac{d_3}{d_1+d_2+d_3}$, $r_1 = 1/r_1, r_2 = 1/r_2, r_3 = 1/r_3$

$$f_i(x_{new}) = r_1\, f_i(x_{old}) + r_2\, f_i(x_{pbest}) + r_3\, f_i(x_{ld}),$$

$i = 1,...,n$. Where f_i is the value of the objective function i and n is the number of objective functions. See Figure 5.3 for an illustration of these techniques.

The technique FI1 is the one proposed in the previous section. As in the previous case, in all the inheritance techniques, if the leader selected does not dominate the current particle, we will proceed as before. See Figure 5.1.

Flight Formula on Objective Space (**FFOS**)

As we mentioned in Chapter 4, in PSO, the position of a particle i in the search space is updated using the formula:

$$\mathbf{x}_i(t) = \mathbf{x}_i(t-1) + \mathbf{v}_i(t)$$

$$\mathbf{v}_i(t) = W\mathbf{v}_i(t-1) + C_1 r_1(\mathbf{x}_{pbest_i} - \mathbf{x}_i(t-1)) + C_2 r_2(\mathbf{x}_{gbest_i} - \mathbf{x}_i(t-1))$$

In this case, we propose the analogous formula to update the position of particle i in the objective space:

$$\mathbf{f}_i(t) = \mathbf{f}_i(t-1) + \mathbf{vf}_i(t)$$
$$\mathbf{vf}_i(t) = W\mathbf{vf}_i(t-1) + C_1 r_1(\mathbf{f}_{pbest_i} - \mathbf{f}_i(t-1)) + C_2 r_2(\mathbf{f}_{gbest_i} - \mathbf{f}_i(t-1))$$

where $\mathbf{f}_i$, $\mathbf{f}_{pbest_i}$ and $\mathbf{f}_{gbest_i}$ are the objective vectors for the current particle i, its *pbest* and *gbest*, respectively. We use the same values of W, C_1, r_1, C_2 and r_2 previously adopted for the flight in the decision variable space. We will consider the following variants based on the vectors considered:

FI4 Considering the whole formula:

$$\mathbf{vf}_i(t) = W\mathbf{vf}_i(t-1) + C_1 r_1(\mathbf{f}_{pbest_i} - \mathbf{f}_i(t-1)) + C_2 r_2(\mathbf{f}_{gbest_i} - \mathbf{f}_i(t-1))$$

FI5 Ignoring the previous direction:

$$\mathbf{vf}_i(t) = C_1 r_1(\mathbf{f}_{pbest_i} - \mathbf{f}_i(t-1)) + C_2 r_2(\mathbf{f}_{gbest_i} - \mathbf{f}_i(t-1))$$

FI6 Ignoring the direction to the *pbest*:

$$\mathbf{vf}_i(t) = W\mathbf{vf}_i(t-1) + C_2 r_2(\mathbf{f}_{gbest_i} - \mathbf{f}_i(t-1))$$

Combination Using Flight Factors

Non-linear Combination (**NLC**)

In this case, we propose to calculate the new objective position of a particle using the elements of the flight formula:

$$\mathbf{f}_i(t) = W\mathbf{f}_i(t-1) + C_1 r_1 \mathbf{f}_{pbest_i} + C_2 r_2 \mathbf{f}_{gbest_i}$$

As in the previous cases, the variants considered are:

FI7 Considering the whole formula:

$$\mathbf{f}_i(t) = W\mathbf{f}_i(t-1) + C_1 r_1 \mathbf{f}_{pbest_i} + C_2 r_2 \mathbf{f}_{gbest_i}$$

FI8 Ignoring the previous position:

$$\mathbf{f}_i(t) = C_1 r_1 \mathbf{f}_{pbest_i} + C_2 r_2 \mathbf{f}_{gbest_i}$$

FI9 Ignoring the position of the *pbest*:

$$\mathbf{f}_i(t) = W\mathbf{f}_i(t-1) + C_2 r_2 \mathbf{f}_{gbest_i}$$

On the other hand, since $W \in (0.1, 0.5)$ and $C_1r_1, C_2r_2 \in (0.0, 2.0)$, we propose to modify the previous formula in the following way:

$$\mathbf{f}_i(t) = \frac{W}{0.5}\mathbf{f}_i(t-1) + \frac{C_1r_1}{2.0}\mathbf{f}_{pbest_i} + \frac{C_2r_2}{2.0}\mathbf{f}_{gbest_i}$$

As a result, we obtain the following variants:

FI10 Considering the whole formula:

$$\mathbf{f}_i(t) = \frac{W}{0.5}\mathbf{f}_i(t-1) + \frac{C_1r_1}{2.0}\mathbf{f}_{pbest_i} + \frac{C_2r_2}{2.0}\mathbf{f}_{gbest_i}$$

FI11 Ignoring the previous position:

$$\mathbf{f}_i(t) = \frac{C_1r_1}{2.0}\mathbf{f}_{pbest_i} + \frac{C_2r_2}{2.0}\mathbf{f}_{gbest_i}$$

FI12 Ignoring the position of the *pbest*:

$$\mathbf{f}_i(t) = \frac{W}{0.5}\mathbf{f}_i(t-1) + \frac{C_2r_2}{2.0}\mathbf{f}_{gbest_i}$$

Linear Combination (**LC**)

We propose to use the previous formula but in such a way that the result is a linear combination of the elements considered:

$$\mathbf{f}_i(t) = \frac{W}{r}\mathbf{f}_i(t-1) + \frac{C_1r_1}{r}\mathbf{f}_{pbest_i} + \frac{C_2r_2}{r}\mathbf{f}_{gbest_i}$$

where $r = W + C_1r_1 + C_2r_2$. The corresponding variants are the following (note the changes in r):

FI13 Considering the whole formula, $r = W + C_1r_1 + C_2r_2$:

$$\mathbf{f}_i(t) = \frac{W}{r}\mathbf{f}_i(t-1) + \frac{C_1r_1}{r}\mathbf{f}_{pbest_i} + \frac{C_2r_2}{r}\mathbf{f}_{gbest_i}$$

FI14 Ignoring the previous position, $r = C_1r_1 + C_2r_2$:

$$\mathbf{f}_i(t) = \frac{C_1r_1}{r}\mathbf{f}_{pbest_i} + \frac{C_2r_2}{r}\mathbf{f}_{gbest_i}$$

FI15 Ignoring the position of the *pbest*, $r = W + C_2r_2$:

$$\mathbf{f}_i(t) = \frac{W}{r}\mathbf{f}_i(t-1) + \frac{C_2r_2}{r}\mathbf{f}_{gbest_i}$$

Fitness Approximation (FA)

We propose four simple approximation techniques. In each case, the particle will take the objective values of the particle indicated:

FA1 The closest particle: leader or member of the swarm.

FA2 The closest leader.

FA3 The closest particle (member of the swarm).

FA4 The average of the 10 closest particles (leaders or members of the swarm).

We use the Euclidean distance in the decision variable space. In technique FA4, there are cases in which an invalid particle may be created. In this way, if among the 10 closest particles there are two or more leaders, or there is just one leader but this leader does not dominate the current particle, we will proceed as it was explained before. See Figure 5.1.

5.4.3 Discussion of Results

The results discussed in this section were published in [79]. We performed a study using functions ZDT1, ZDT2, ZDT3 and ZDT4 (previously defined). Also, we used different values of *inheritance (approximation) proportion* p_i, we experimented with: p_i= 0.1, 0.2, 0.3, 0.4. We performed 20 runs for each function and each technique. The parameters adopted for our MOPSO were: 100 particles, 200 generations and 100 particles in the external archive. We show the results corresponding to the Success Counting (SCC) measure (previously defined). Tables C.4, C.5, C.6 and C.7 present a summary of the results obtained, and can be consulted in Appendix C, page 191. In each case, we present the average of the SCC measure over the 20 runs, and the percentage of decrement or increment on the quality of the results. Also, we present the average of the percentages for each value of inheritance proportion, for each technique.

Since comparing 19 different techniques is very difficult, we decided to represent each technique with a vector. The vector used is that containing the average of the change in the quality of results for each inheritance proportion value. For example, to represent technique FI1, we construct the following vector (see Table C.4):

Inheritance proportion p_i	0.1	0.2	0.3	0.4
Average vector	2.6	-4.1	-13.7	-14.0

In this way, in Table 5.1 we present the vectors of all techniques. Since every entry in each vector is a change in the quality of the obtained results given a value of inheritance proportion, the bigger the values of the vector, the better the corresponding technique is. Thus, we are interested on the vector or vectors that represent the solution to the problem of maximizing all the entries (i.e. each entry is considered as an objective). The non-dominated vectors among all the 19 techniques are the vectors corresponding to techniques FI5 and FA3. That is, the techniques FI5 and FA3 are the best. For this reason these two techniques are marked with a level of 1 in Table 5.1. FI5 is an inheritance technique and FA3 is an approximation technique. For these two techniques, in the worst case, the decrement in quality of results is no more than 13%, even when a 40% of the total number of evaluations is saved. After eliminating techniques FI5 and FA3, we proceed again to locate the non-dominated vectors. In this case, the best techniques, marked with a level 2 are: FI1, FI2, FI4, FI6 and FA1. This leads us to conclude that, in general, the set of inheritance techniques based on the flight formula on the objective space (FFOS) are the best.

5.4.4 Comparison with other PSO approaches

In the previous section, we found two enhancement techniques to be the best from the set proposed: one of fitness inheritance and one of fitness approximation. In this section, these two techniques are compared against other two PSO-based multi-objective approaches representative of the state-of-the-art: the Sigma-MOPSO [68] and the Cluster-MOPSO [103]. For this comparison we use two different test functions: DTLZ2 and DTLZ6 (previously defined). As in previous experiments, we used different values of p_i. We performed 20 runs for each function and each approach. The approaches without fitness inheritance or approximation performed 20000 objective function evaluations. The PSO approaches will be identified with the following labels: sMOPSO refers to [68], cMOPSO refers to [103], and oMOPSO is our MOPSO. All the algorithms were set such that they provided Pareto fronts with 100 points. In this case, we also show the obtained results with respect to the Inverted Generational Distance (IGD) measure (previously defined).

Tables C.8 and C.9 present a summary of the results obtained (Appendix C, page 191). In each case, we present the average and standard deviation of the SCC and IGD measures over the 20 runs. From Table C.8, we conclude that the fitness inheritance technique FI5 has a better performance in function DTLZ6 than in function DTLZ2 with respect to the SCC measure. However, the IGD

Group		0.1	0.2	0.3	0.4	level
LCBD	FI1	2.6	-4.1	-13.7	-14.0	**2**
	FI2	-3.6	-2.4	-11.9	-12.9	**2**
	FI3	0.1	-4.9	-13.8	-17.8	
FFOS	FI4	0.1	-1.7	-8.7	-13.6	**2**
	FI5	**4.7**	**-1.2**	**-8.1**	**-11.7**	**1**
	FI6	1.6	-2.8	-10.1	-16.7	**2**
NLC	FI7	-4.9	-10.3	-19.5	-30.2	
	FI8	-0.7	-7.5	-20.7	-29.8	
	FI9	-0.2	-7.3	-16.7	-28.5	
	FI10	-3.0	-9.2	-19.3	-33.3	
	FI11	-3.6	-6.0	-14.1	-26.7	
	FI12	-3.0	-9.5	-17.5	-22.1	
LC	FI13	-2.1	-2.6	-12.5	-18.8	
	FI14	-3.7	-4.9	-10.3	-16.0	
	FI15	0.3	-5.0	-12.3	-16.6	
FA	FA1	4.2	-3.4	-8.4	-14.1	**2**
	FA2	-0.3	-11.2	-16.6	-15.9	
	FA3	**1.5**	**0.4**	**-6.9**	**-12.9**	**1**
	FA4	0.3	-4.1	-12.3	-16.2	

Table 5.1: Vectors of change in quality for each technique, for each value of inheritance or approximation proportion.

measure indicates a very good performance in all cases, even with respect to the other PSO-based approaches. Table C.9 shows a very good performance of the fitness approximation technique FA3 in both functions and with respect to the two measures.

In general, technique FA3 was better than FI5 in function DTLZ2, in which it offers a 12% of decrement in quality with a saving of 30% in evaluations in the best case, and a 28% of decrement in quality with a saving of 40% in evaluations, in the worst case. On the other hand, technique FI5 was better than FA3 in function DTLZ6. In function DTLZ6, technique FI5 offers a 2% of decrement in quality with a saving of 30% in evaluations in the best case, and a 30% of decrement in quality with a saving of 40% in evaluations, in the worst case. These results agree with those obtained before. As we can see in Table 5.2, the results obtained in the previous study show that technique FA3 is consistently better than technique FI5 in function ZDT4. In this way, we can conclude that the fitness approximation

FI5	Inheritance proportion p_i								
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)		0.4 (-40%)	
ZDT1	71	**74**	(+4.2%)	**69**	(-2.8%)	61	(-14.1%)	**56**	(-21.1%)
ZDT2	89	**89**	(0.0%)	79	(-11.2%)	84	(-5.6%)	**77**	(-13.5%)
ZDT3	68	**72**	(+5.9%)	**70**	(+2.9%)	**55**	(-19.1%)	**58**	(-14.7%)
ZDT4	80	87	(+8.8%)	85	(+6.3%)	85	(+6.3%)	82	(+2.5%)
Average			**+4.7%**		**-1.2 %**		**-8.1 %**		**-11.7%**
FA3	Approximation proportion p_a								
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)		0.4 (-40%)	
ZDT1	71	71	(0.0%)	67	(-5.6%)	**63**	(-11.3%)	50	(-29.6%)
ZDT2	89	88	(-1.1%)	**87**	(-2.2%)	**85**	(-4.5%)	76	(-14.6%)
ZDT3	68	65	(-4.4%)	65	(-4.4%)	55	(-19.1%)	57	(-16.2%)
ZDT4	80	**89**	(+11.3%)	**91**	(+13.8%)	**86**	(+7.5%)	**87**	(+8.8%)
Average			**+1.5%**		**+0.4 %**		**-6.9 %**		**-12.9%**

Table 5.2: Results obtained for different values of inheritance and approximation proportion, for techniques FI5 and FA3.

technique FA3 has better results when the test function has a low dimensional decision space and that the fitness inheritance technique FI5 has better results when the test function has a high dimensional decision space. This conclusion seems to agree with the results obtained from the first fitness inheritance technique proposed.

From the nineteen techniques proposed to reduce the computational cost, we found two of them as the best: one inheritance technique and one approximation technique. The best fitness inheritance technique is based on the flight formula for the objective space proposed by us. The best approximation technique is based on the simple idea of assigning to a particle the same objective values of the closest particle member of the swarm. Both techniques were tested on other functions and compared with other PSO-based multi-objective algorithms. The obtained results show that both enhancement techniques have a good performance and are very promising.

On the other hand, fitness inheritance techniques seem to be more appropriate for high-dimensional decision space problems and fitness approximation techniques seem more appropriate for low-dimensional decision space problems.

Since our major interest is reducing computational cost, in the next section we describe the work done in order to improve the results described until now.

5.5 Dynamic Inheritance Proportion

In this chapter, we have shown that the use of an enhancement technique is very useful to reduce the computational cost of EAs.

The fitness inheritance (and approximation) techniques proposed have reduced the computational cost significantly without decreasing the quality of the results in a dramatic way. However, in all the previous experiments, the savings in the number of evaluations has been completely determined by the value of inheritance (or approximation) proportion p_i.

With the aim of obtaining major savings in the number of evaluations performed, we decided to study the possibility of setting the value of the inheritance proportion parameter p_i following a dynamical scheme.

5.5.1 Proposed Dynamical Approach

As it has been mentioned before, the use of fitness inheritance decreases the quality of the results as we increase the value of the parameter p_i. So, our main idea is to increase the saving in number of function evaluations but setting the value of p_i in such a way that the quality of the obtained results can be less affected.

We proceeded to analyze the behavior of our MOPSO approach, with respect to the improvement on the results through the evolutionary process, that is, through the generations.

Given the current Pareto front in generation t, $\mathrm{PF}(t)$, and the Pareto front in the previous generation $\mathrm{PF}(t-1)$, we calculated the value of the Two Set Coverage measure SC (defined in Section 2.4.4, page 25):

$$\mathrm{SC}(\mathrm{PF}(t), \mathrm{PF}(t-1))$$

In this way, we can know "how much" better is the current Pareto front with respect to the front of the previous generation.

We performed 20 runs of our approach using function ZDT1, 100 particles and 200 generations, and we obtained the average of $\mathrm{SC}(\mathrm{PF}(t),\mathrm{PF}(t-1))$ on each generation. Figure 5.4 shows the obtained results.

As we can see in Figure 5.4, the most important improvement takes place during the first quarter of the total of generations (the first 50 generations in this case). In this way, we concluded that at the beginning of the process it is not convenient to use too much fitness inheritance. However, towards the end of the process, fitness inheritance is more suitable.

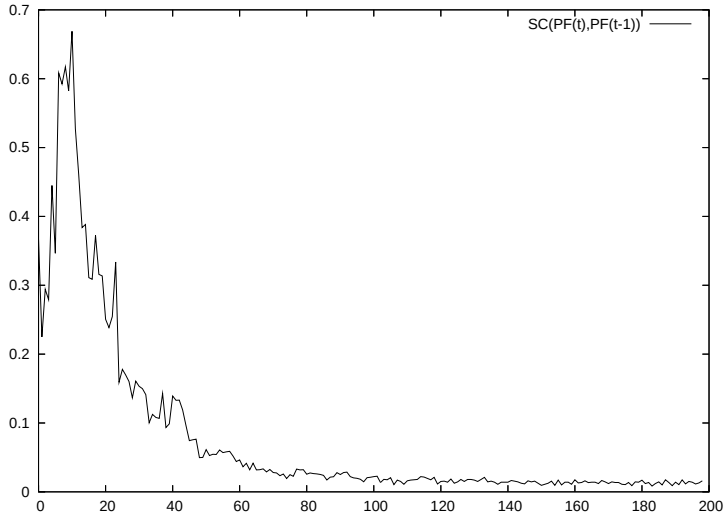


Figure 5.4: Two Set Coverage measure (SC).

Thus, given the previous conclusions, we propose to set the value of the parameter p_i dynamically with respect to the current generation number. The main idea is to increase the use of fitness inheritance through the evolutionary process. In this way, we propose six different functions to adapt the value of the inheritance proportion with respect to the number of the current generation: Let *gen* be the number of the current generation and *Gmax* the total number of generations:

- nonlinear1: $p_i(gen) = \left(\frac{gen}{Gmax}\right)^4$
- nonlinear2: $p_i(gen) = \left(\frac{gen}{Gmax}\right)^2$
- nonlinear3: $p_i(gen) = \frac{gen}{Gmax} - \frac{\mathtt{sin}(2\Pi\frac{gen}{Gmax})}{6.3}$
- linear: $p_i(gen) = \frac{gen}{Gmax}$
- nonlinear4: $p_i(gen) = \left(\frac{gen}{Gmax}\right)^{\frac{1}{2}}$
- nonlinear5: $p_i(gen) = \left(\frac{gen}{Gmax}\right)^{\frac{1}{4}}$

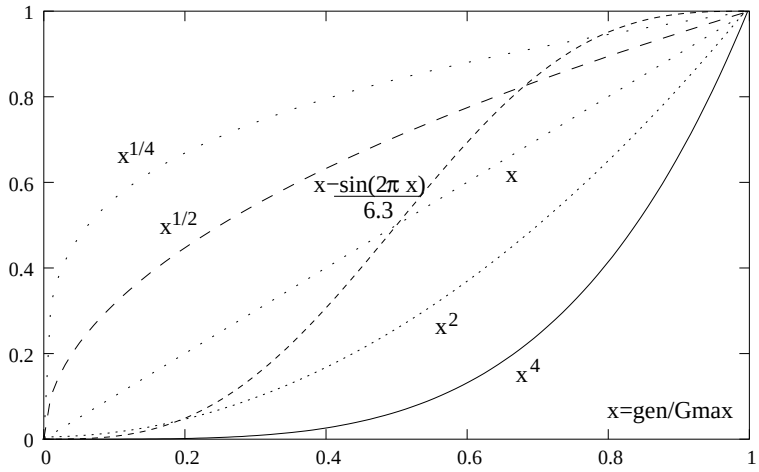


Figure 5.5: Plot of the six different functions proposed to adapt the value of the inheritance proportion (p_i) through the evolutionary process.

The six previous functions (that we will call *adaptive functions*) are designed in such a way that the value of inheritance proportion increases through the evolutionary process. Nevertheless, it is important to note that $gen = 0, \ldots, Gmax - 1$, and that $p_i = 0$ when $gen = 0$, since fitness inheritance must not be applied when $gen = 0$. Figure 5.5 presents a plot of the six adaptive functions.

The adaptive functions are numbered following an ascending order with respect to the savings on the total number of evaluations that they define. In this way, the adaptive function nonlinear1 is the one that defines the least savings on the total number of evaluations. On the other hand, the adaptive function nonlinear5 is the one that defines the greatest savings on the total number of evaluations. See Figure 5.5.

5.5.2 Discussion of Results

The results discussed in this section were published in [80], and can also be seen in [81]. In order to test the mechanism proposed to set the value of the parameter p_i, we proceeded to incorporate it into the MOPSO approach and we performed the following experiments. We ran our algorithm 30 times using functions ZDT1, ZDT2, ZDT3, ZDT4 and DTLZ6 (previously defined).

The parameters used were 200 particles, 100 generations and 100 points in

the final Pareto Front. In this way, the total number of evaluations is 20200 in the absence of fitness inheritance. In all the experiments performed we used the fitness inheritance technique FI5, since it was found to be the best among the inheritance techniques previously described in Section 5.4. For our comparative study, we used the SCC and IGD measures of performance (previously defined). All the tables and figures corresponding to the obtained results can be consulted in Appendix C, page 191.

Tables C.10, C.11 and C.12 present the results obtained using the unary measures. For each test function, we present first the results obtained by the MOPSO approach without inheritance and then, the results of the approach with inheritance with each one of the adaptive functions, from function nonlinear1 to function nonlinear5. For each approach, we present the best, median, worst, mean and standard deviation values with respect to the two unary measures implemented. Also, we show the average number of evaluations performed on each case and the corresponding percentage of savings obtained.

On the other hand, Tables C.13, C.14 and C.15 present the confidence intervals for the mean statistic (with 95% of confidence), for the SCC and IGD measures, for each function and each approach. We performed a Kolmogorov-Smirnov test for each sample of each function and each approach, to verify if the corresponding distributions were close to a normal one. For those distributions that were not close to a normal, we calculated the bootstrap bias-corrected accelerated (BCA).

In addition, for each test function, we selected the Pareto fronts corresponding to the median value with respect to the SCC measure to represent each approach. In this way, Table C.16 presents the results obtained when applying the SC measure to the Pareto fronts that represent each approach. For each case, we compare the Pareto front corresponding to the approach without inheritance against the Pareto fronts corresponding to the approaches with inheritance, using adaptive functions nonlinear1 to nonlinear5. Also, for each case we compare the Pareto fronts corresponding to the approaches with inheritance against the Pareto front of the approach without inheritance.

Finally, Figures C.3, C.4 and C.5 show the Pareto fronts that represent each approach. For each test function, we present three plots. The first one, shows the Pareto fronts from the approach without inheritance and the approaches with inheritance and adaptive functions nonlinear1 and nonlinear2. The second plot shows the Pareto fronts from approaches with inheritance and adaptive functions nonlinear3 and linear. Finally, the third plot shows the Pareto fronts from approaches with inheritance and adaptive functions nonlinear4 and nonlinear5.

As we can see in Tables C.10 to C.15, for all the test functions, when the

application of the inheritance increases, the quality on the results is more affected. This behavior can be observed more dramatically in functions ZDT1, ZDT3 and DTLZ6. In functions ZDT2 and ZDT4, the quality of the results is less affected by the application of fitness inheritance. In fact, in function ZDT4, we can see that the results with respect to the IGD measure are almost of the same quality, even when a 78% of evaluations are saved. Actually, in function ZDT4 we argue that, considering both measures of performance, it is possible to save even a 49% of function evaluations without affecting the quality of the results.

Taking into account both unary measures of performance, we consider that the quality of the results is maintained when at least one of the measures indicates so. In this way, we can conclude that in general, it is possible to save even a 32% of evaluations without affecting the quality of the obtained solutions. That is, adaptive functions nonlinear1 and nonlinear2, seem to be good options to save evaluations without deteriorating the quality of the results.

On the other hand, as it was expected from their definition, adaptive functions nonlinear3 and linear, provide the same percentage of savings in the number of evaluations. In general, the results when using function linear are better than the corresponding results using function nonlinear3. Since both adaptive functions (nonlinear3 and linear) provide the same amount of savings, we conclude that given their corresponding definition, the obtained results indicate that it is important to maintain the true evaluations at the end of the run, even if we try not to apply inheritance at the beginning of the evolutionary process.

As we can see in Tables C.10 to C.15, adaptive functions that save more than a 50% of the total number of evaluations, nonlinear4 and nonlinear5, affect the results more dramatically. That is, the damage on the quality of the results increase more rapidly when the savings are greater than a 50% of the evaluations.

Very similar conclusions can be obtained from the results obtained using measure SC. In Table C.16, we can see again that the percentage of dominated solutions by the Pareto front corresponding to the approach without inheritance grows when we increase the use of fitness inheritance. In functions ZDT2 and ZDT4, the quality of the results is less affected by the use of inheritance, especially in the case of function ZDT4. In the case of adaptive functions nonlinear1 and nonlinear2, the results in Table C.16 indicate that the Pareto fronts obtained using function nonlinear2 are better. On the other hand, results from adaptive functions nonlinear3 and linear are very similar, but we can conclude again that function linear provides better results than function nonlinear3. Finally, it is very clear that the use of adaptive functions nonlinear4 and nonlinear5 affects dramatically the obtained results, especially in the case of functions ZDT1 and ZDT3.

From Table C.16, we can conclude that it may be possible to save a 65% of the total number of evaluations expecting to lose a 40% of quality, in the worst case. Also, it seems very harmful to save a 78% of evaluations, since in the worst case the quality was reduced by 67%.

Finally, as we can see in Figures C.3, C.4 and C.5, the Pareto fronts generated by the approaches with inheritance are very similar to the Pareto front from the approach without inheritance, even when a 49% of the total number of evaluations is saved. It is only in the case of the approaches with inheritance and adaptive functions nonlinear4 and nonlinear5 in which we can observe Pareto fronts of relatively low quality (especially in the case of function DTLZ6 and adaptive function nonlinear5). However, we can see that the Pareto fronts obtained by those approaches are very good approximations of the true Pareto front, except in the case of function DTLZ6, in which adaptive function nonlinear5 lost some portions of the true Pareto front.

In general, from the obtained results shown in Tables C.10 to C.16, we can conclude that it is possible to save a 32% of the total number of evaluations without significantly affecting the quality of the obtained solutions. Also, the quality of the results when having savings of 49% of the evaluations, is very acceptable. Finally, in the case of reducing the total number of evaluations by more than a 50%, the quality of the results is even more affected. However, as we can see in the Pareto fronts shown, even with a 78% of savings, the approach with inheritance still provides very good approximations of the true Pareto front. In this way, although the quality of the results can be significantly affected when reducing the computational cost by more than 50%, if the real world application is very expensive to evaluate and we are interested only on a few optimal solutions, the proposed approach may be a suitable choice.

5.5.3 Comparison with Other MOEAs

In this section, we present a final comparison of the results discussed in the previous section, with respect to the results of the NSGA-II and SPEA2 algorithms, provided in Chapter 4.

In Table 5.3, we show the results of the inheritance approach previously discussed, and the results of the NSGA-II and the SPEA2, for functions ZDT1, ZDT2, ZDT3, ZDT4 and DTLZ6. We show in boldface the cases in which the results of our approach are either better or of the same quality (on average) than the corresponding results of both the NSGA-II and the SPEA2.

As we can see in Table 5.3, in functions ZDT2 and ZDT4 our results are

		OMOPSO							NSGA-II	SPEA2
savings		**0%**	**19%**	**32%**	**49%**	**49%**	**65%**	**78%**	**0%**	**0%**
ZDT1										
SCC	mean	**87**	**84**	**74**	**71**	**68**	**53**	21	21	27
	st.dv.	12.5	12.6	21	18.6	22.7	21.6	13.5	7.5	8.1
IGD	mean	**0.0010**	**0.0010**	**0.0010**	0.0031	0.0028	0.0039	0.0084	0.0009	0.0007
	st.dv.	0.0000	0.0000	0.0002	0.0089	0.0069	0.0098	0.0180	0.0001	0.0001
ZDT2										
SCC	mean	**92**	**93**	**89**	**83**	**84**	**69**	**45**	6	7
	st.dv.	12.9	6.1	12.2	21.7	22.9	26.6	34.2	9.8	10.4
IGD	mean	**0.0007**	**0.0007**	**0.0007**	**0.0009**	**0.0008**	**0.0052**	**0.0038**	0.0512	0.0404
	st.dv.	0.0001	0.0000	0.0000	0.0010	0.0005	0.0139	0.0090	0.0337	0.0367
ZDT3										
SCC	mean	**76**	**73**	**72**	**53**	**59**	37	16	44	39
	st.dv.	12.7	11.6	15.9	21.5	16.2	18	12.6	6.8	6.0
IGD	mean	**0.0009**	**0.0010**	**0.0020**	0.0074	0.0023	0.0109	0.0178	0.0013	0.0018
	st.dv.	0.0001	0.0003	0.0032	0.0138	0.0049	0.0137	0.0147	0.0021	0.0030
ZDT4										
SCC	mean	**96**	**94**	**93**	**89**	**90**	**77**	**47**	0	0
	st.dv.	4.8	6.6	6.0	12.6	14.2	18.1	22.6	0	0
IGD	mean	**0.0010**	**0.0010**	**0.0010**	**0.0010**	**0.0010**	**0.0010**	**0.0012**	0.1508	0.1224
	st.dv.	0.0000	0.0000	0.0000	0.0000	0.0000	0.0001	0.0003	0.0973	0.0943
DTLZ6										
SCC	mean	**74**	**71**	**66**	**52**	**52**	**42**	**20**	0.1	0.6
	st.dv.	14.2	15.8	14.2	14.6	19.2	14.3	7.9	0.3	0.9
IGD	mean	**0.0064**	**0.0087**	**0.0104**	0.0126	0.0118	0.0139	0.0205	0.0132	0.0067
	st.dv.	0.0053	0.0057	0.0053	0.0048	0.0067	0.0061	0.0082	0.0083	0.0051

Table 5.3: Comparison of the results obtained by the inheritance approach, with respect to the NSGA-II and the SPEA2.

better than the results of the NSGA-II and the SPEA2, even when a 78% of the total number of evaluations is saved. In the rest of the functions, we can see that we are able to save at least a 49% of the total number of evaluations, while obtaining results of better quality than the two other MOEAs, with respect to the SCC measure. Also, with respect to the IGD measure, we are able to save at least a 32% of the total number of evaluations, without affecting the quality of the results in a significant way.

In this way, we can see that the proposed fitness inheritance approach enables our MOPSO algorithm to obtain better results than the NSGA-II and the SPEA2 (which are two of the most representative algorithms of the state-of-the-art), while performing a smaller number of function evaluations.

5.6 Conclusions

In this chapter, we first propose the use of fitness inheritance in order to reduce the computational cost of the PSO-based multi-objective approach described in the previous chapter. Since the obtained results with the first technique proposed were very promising, we study several different ways of incorporating such enhancement technique and also experimented with some simple fitness approximation schemes.

Since both fitness inheritance and approximation techniques provide a reduction in the computational cost that is completely determined by the value of inheritance (or approximation) proportion p_i, we proposed mechanisms to adapt the value of the inheritance proportion in a dynamical way, throughout the evolutionary process. Six different functions to adapt the inheritance proportion were proposed, each one defining a different percentage of savings, from 19% to 78% of the total number of evaluations.

From the results obtained, we can conclude that, in general, the use of fitness inheritance affects the quality of the provided Pareto fronts as we increase the number of true evaluations saved. However, such effect was less noticeable in the test function with the decision space of lowest dimensionality. On the other hand, we conclude that it is possible to save until a 32% of the total number of evaluations without significantly deteriorating the quality of the results. When the proposed approaches save a 49% of evaluations, the effect on the quality of the results is more noticeable. However, the provided results are of very good quality. Furthermore, although the quantitative quality of the Pareto fronts provided by the approaches that save 65% and 78% of evaluations is more affected, the corre-

sponding plots show that such strategies are able to generate good approximations of the true Pareto front.

Finally, when comparing the results provided by the proposed fitness inheritance approach with respect to two of the most representative algorithms of the state-of-the-art, the NSGA-II and the SPEA2, we conclude that the proposed mechanism enables the MOPSO approach previously described, to obtain better results while performing a smaller number of function evaluations.

Chapter 6

Theoretical Issues

In this chapter, we discuss some theoretical issues related to the work developed in this thesis, about PSO and fitness inheritance.

With the aim of exploring the convergence properties of the MOPSO approach described in this thesis, we first discuss the convergence properties of PSO. We briefly describe some of the characteristics of the models that have been developed to analyze the convergence of PSO and present an outline of the model which is more related with our MOPSO approach. Afterwards, we obtain some conclusions about the convergence properties of our algorithm.

For the case of the study of fitness inheritance, we proceed in a similar way. We first present the existing theoretical studies about fitness inheritance. We present the models developed by Sastry [91] and Chen [12], which analyze the behavior of fitness inheritance when it is applied into a genetic algorithm, for the single and multi-objective cases of a very simple problem, respectively. As we will see, these models are based on the use of binary representation and some other hypothesis which make them difficult to use in order to explore the properties of the fitness inheritance techniques proposed as part of our work.

In this way, in the last part of this chapter, we provide a simple analysis of the behavior of one of the fitness inheritance techniques presented in Chapter 5. We discuss the impact of the fitness inheritance technique studied on the trajectory of the particles of the swarm, when it is applied to real-coded vectors and under the mechanisms defined by our MOPSO approach. With this analysis, we attempt to explore the possible reasons of the effect of fitness inheritance in the quality of the results, previously observed. Some interesting conclusions are obtained as well as some directions for future research.

6.1 Convergence Properties of PSO and MOPSO

Recently, some theoretical studies about the convergence properties of PSO have been published. As in the case of many evolutionary algorithms, these studies have concluded that the performance of the PSO is sensitive to control parameter choices [32].

Most of the theoretical studies are based on simplified PSO models, in which the trajectory of one particle of one dimension is studied, assuming that the other particles in the swarm will remain "frozen" while the trajectory is analyzed. The *pbest* and *gbest* particles are assumed to be constant throughout the process. Also, the terms $\phi_1 = C_1 r_1$, $\phi_2 = C_2 r_2$ (used in Equation 4.2) are assumed to be constant. Under these conditions, particle trajectories and convergence of the swarm have been analyzed.

In the theoretical studies developed about PSO, convergence has been defined as follows:

Definition 1. Considering the sequence of global best solutions $\{gbest_t\}_{t=0}^{\infty}$, we say that the swarm converges iff

$$\lim_{t\to\infty} gbest_t = p$$

where p is an arbitrary position in the search space.

Since p refers to an arbitrary solution, Definition 1 does not mean convergence to a local or global optimum.

The first studies on the convergence properties of PSO were developed by Ozcan and Mohan [71, 72]. Ozcan and Mohan studied a PSO under the conditions previously described but, in addition, their model did not consider the inertia weight. They concluded that, when $0 < \phi < 4$, where $\phi = \phi_1 + \phi_2$, the trajectory of a particle is a sinusoidal wave where the initial conditions and parameter choices determine the amplitude and frequency of the wave. Also, they concluded that the periodic nature of the trajectory may cause a particle to repeatedly search regions of the search space already visited, unless another particle in its neighborhood finds a better solution.

In [104], van den Bergh developed a model of PSO under the same conditions, but considering the inertia weight. We present here the model developed by van den Bergh.

Using $\phi_1 = C_1 r_1$ and $\phi_2 = C_2 r_2$, we repeat here Equations 4.1 and 4.2, corresponding to the case of a particle x of one dimension:

$$v(t+1) = Wv(t) + \phi_1(pbest - x(t)) + \phi_2(gbest - x(t)) \tag{6.1}$$

$$x(t+1) = x(t) + v(t+1) \tag{6.2}$$

where *pbest* is the best position found by particle x and *gbest* is the best position found by the entire swarm (which are supposed to remain constant).

Substituting 6.1 into 6.2, the following recurrence relation is obtained:

$$x_{t+1} = (1 + W - \phi_1 - \phi_2)x_t - W x_{t-1} + \phi_1 pbest + \phi_2 gbest \tag{6.3}$$

where x_t denotes $x(t)$. This recurrence relation can be written as a matrix-vector product, so that

$$\begin{bmatrix} x_{t+1} \\ x_t \\ 1 \end{bmatrix} = \begin{bmatrix} 1 + W - \phi_1 - \phi_2 & -W & \phi_1 pbest + \phi_2 gbest \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ x_{t-1} \\ 1 \end{bmatrix} \tag{6.4}$$

The characteristic polynomial of the matrix 6.4 is:

$$(1-\lambda)(W - \lambda(1 + W - \phi_1 - \phi_2) + \lambda^2)$$

which has a trivial solution of $\lambda = 1.0$ and two other solutions

$$\alpha = \frac{1 + W - \phi_1 - \phi_2 + \gamma}{2}$$

$$\beta = \frac{1 + W - \phi_1 - \phi_2 - \gamma}{2}$$

where

$$\gamma = \sqrt{(1 + W - \phi_1 - \phi_2)^2 - 4W}$$

The explicit form of the recurrence relation 6.3 is then given by

$$x(t) = k_1 + k_2\alpha^t + k_3\beta^t \tag{6.5}$$

where k_1, k_2 and k_3 are constants determined by the initial conditions of the system. Since there are three unknowns, a system of three equations must be constructed to find their values. The initial conditions of PSO provide two such conditions, x_0 and x_1, corresponding to the position of the particle at time steps 0 and 1. The third constraint of the system can be calculated using the recurrence relation to find the value of x_2, thus,

$$x_2 = (1 + W - \phi_1 - \phi_2)x_1 - Wx_0 + \phi_1 pbest + \phi_2 gbest$$

From these initial conditions, the system

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \alpha & \beta \\ 1 & \alpha^2 & \beta^2 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix}$$

is derived. Using the property $\alpha - \beta = \gamma$, these equations can be further simplified to yield

$$k_1 = \frac{\phi_1 pbest + \phi_2 gbest}{\phi_1 + \phi_2}$$

$$k_2 = \frac{\beta(x_0 - x_1) - x_1 + x_2}{\gamma(\alpha - 1)}$$

$$k_3 = \frac{\alpha(x_1 - x_0) + x_1 - x_2}{\gamma(\beta - 1)}$$

Equation 6.5 can be used to compute the trajectory of a particle, under the assumption that *pbest*, *gbest*, ϕ_1, ϕ_2 and W remain constant. Convergence of the sequence $\{x_t\}_{t=0}^{\infty}$ is determined by the magnitude of the values α and β. In this way, when $\max\{\|\alpha\|, \|\beta\|\} < 1$ we have:

$$\lim_{t\to\infty} x_t = \lim_{t\to\infty} k_1 + k_2\alpha^t + k_3\beta^t = k_1$$

Thus

$$\lim_{t\to\infty} x_t = \frac{\phi_1 pbest + \phi_2 gbest}{\phi_1 + \phi_2} \tag{6.6}$$

If the expected values of ϕ_1 and ϕ_2:

$$E[\phi_1] = C_1 \int_0^1 x dx = \frac{C_1}{2}$$

$$E[\phi_2] = C_2 \int_0^1 x dx = \frac{C_2}{2}$$

are substituted in 6.6, we obtain:

$$\lim_{t\to\infty} x_t = \frac{C_1 pbest + C_2 gbest}{C_1 + C_2}$$

In this way, if $C_1 = C_2$, the particle converges to the point

$$\frac{pbest + gbest}{2}.$$

In general, for arbitrary values of C_1 and C_2, we have:

$$\begin{aligned} \lim_{t\to\infty} x_t &= \frac{C_1 pbest + C_2 gbest}{C_1 + C_2} \\ &= \frac{C_1}{C_1 + C_2} pbest + \frac{C_1}{C_1 + C_2} gbest \\ &= (1 - \frac{C_2}{C_1 + C_2}) pbest + \frac{C_1}{C_1 + C_2} gbest \\ &= (1 - a) pbest + agbest \end{aligned}$$

where $a = \frac{c_2}{c_1 + c_2}$. In this way, van den Bergh proved that a particle converges to a weighted average between its personal best and its neighborhood best position.

As we said before, the condition $\max\{\|\alpha\|, \|\beta\|\} < 1$ is needed to ensure convergence. In [104], van den Bergh experimentally concluded that such condition is equivalent to have:

$$w > \frac{1}{2}(c_1 + c_2) - 1 \tag{6.7}$$

However, van den Bergh also noted that it is not strictly necessary to choose the values of C_1, C_2 and W, so that relation 6.7 is satisfied. Considering the stochastic nature of C_1 and C_2, it is possible to choose values of C_1, C_2 and W such that condition 6.7 is violated, and the swarm still converges [104]: if

$$\phi_{ratio} = \frac{\phi_{crit}}{c_1 + c_2}$$

is close to 1.0, where $\phi_{crit} = \sup \{\phi \mid 0.5\phi - 1 < w\}$, $\phi \in (0, c_1 + c_2]$, the swarm has convergent behavior. This implies that the trajectory of the particle will converge *most of the time*, occasionally taking divergent steps.

The studies developed by Ozcan and Mohan, and van der Bergh, consider trajectories that are not constricted. In [15], Clerc and Kennedy provide a theoretical analysis of particle behavior in which they introduce a constriction coefficient whose objective is to prevent the velocity from growing out of bounds.

As we could see, the convergence of PSO has been proved. However, we can only ensure the convergence of PSO to the best position visited by all the particles

of the swarm. In order to ensure convergence to the local or global optimum, two conditions are necessary [86, 104]:

(1) The $gbest_{t+1}$ solution can be no worse than the $gbest_t$ solution (monotonic condition).

(2) The algorithm must be able to generate a solution in the neighborhood of the optimum with nonzero probability, from any solution x of the search space.

In [104], van den Bergh provides a proof to show that the basic PSO is not a local (neither global) optimizer. This is due to the fact that, although PSO satisfies the monotonic condition indicated above, once the algorithm reaches the state where $x = pbest = gbest$ for all particles in the swarm, no further progress will be made. The problem is that this state may be reached before *gbest* reaches a minimum, whether be local or global. The basic PSO is therefore said to prematurely converge. In this way, the basic PSO algorithm is not a local (global) search algorithm, since it has no guaranteed convergence to a local (global) minimum from an arbitrary initial state.

Also, van den Bergh suggests two ways of extending PSO in order to make it a global search algorithm. The first is related to the generation of new random solutions. In general, the introduction of a mutation operator is useful. Nevertheless, forcing PSO to perform a random search in an area surrounding the global best position, that is, forcing the global best position to change in order to prevent stagnation (by means of a hill-climbing search, for example), is also a suitable mechanism [32]. On the other hand, van den Bergh also proposes to use a "multi-start PSO", in which, when the algorithm has converged (under some criteria), it records the best solution found and the particles are randomly reinitialized.

To the best of our knowledge, until this date, there are no studies about the convergence properties of MOPSOs. From the discussion previously provided, we can conclude that it is possible to ensure convergence, by correctly setting the parameters of the flight formula. But, as in the case of single-objective optimization, such property does not ensure the convergence to the true Pareto front, in this case. In the case of multi-objective optimization, we may conclude that we still need conditions (1) and (2), to ensure convergence. However, in this case, condition (1) may change to:

(1') The solutions contained in the external archive at iteration $t+1$ should be nondominated with respect to the solutions generated in all iterations τ, $0 \leq \tau \leq t+1$, so far (monotonic condition).

The use of the ε-dominance based archiving, as proposed in [58] ensures this condition, but the normal dominance-based strategies do not, unless they incorporate a mechanism to ensure that for any solution discarded from the archive one with equal or dominating objective vector is accepted. In this way, given a MOPSO approach, and assuming it satisfies condition (1), it remains to explore if it satisfies condition (2), to ensure global convergence to the true Pareto front.

In the particular case of the MOPSO approach proposed in this work, as described in Chapter 4, we have that it satisfies conditions (1') and (2). Condition (1') is satisfied since our MOPSO approach uses the ε-dominance based archiving mechanism which, as said before, ensures condition (1'). On the other hand, one of the mutation operators used by our MOPSO approach ensures condition (2). The uniform mutation operator is able to generate any solution of the search space from any solution x with nonzero probability, including those solutions in the neighborhood of the Pareto optimal solutions [86, 87]. In this way, we can conclude that our MOPSO approach converges to the true Pareto front.

6.2 Fitness Inheritance for the OneMax Problem

The existing theoretical studies about fitness inheritance are those developed by Sastry et al. [91] and Chen et al. [12]. They investigated convergence times, population sizing and the optimal proportion of inheritance for the OneMax problem, for the single and multi-objective case, respectively.

In this section, we first present the model developed by Sastry, for the single objective case, and then we present the extensions performed by Chen, for the multi-objective case. The material presented in this section was taken from the masters [90] and PhD [11] theses of Sastry and Chen, respectively.

The following two definitions are useful for the understanding of the material discussed here.

Definition 2. A *schema* $H = (h_1, h_2, \ldots, h_l)$ is defined as a string of lenght l, where $h_i = \{0, 1, *\}$, $i = 1, 2, \ldots, l$ and $*$ denotes the "don't care" symbol.

Definition 3. If a solution A of a problem is an instance of a schema H, the schema H is called a *building block (BB)*.

The OneMax problem is well-known and well-studied in the context of GAs. The OneMax problem, originally defined as single-objective, is a bit-counting problem where the fitness value of a binary string is equal to the number of 1s in

it. The OneMax problem is defined as follows:

$$f = \sum_{i=1}^{l} x_i$$

where x_i is the value of the ith bit. The global optimum of the OneMax problem is a string with all 1s and its fitness value is equal to the string length l. The OneMax problem has the property that the fitness is binomially distributed. Therefore, the mean of fitness values can be written as

$$\mu_t = l p_t \tag{6.8}$$

and the variance as

$$\sigma_t^2 = l p_t (1 - p_t) \tag{6.9}$$

where p_t is the proportion of correct BBs at generation t. On the other hand, the OneMax is a uniformly scaled problem, that is, the contribution of fitness by building blocks from different partitions are the same. This equal salience of building blocks leads to a parallel search in terms of the best building blocks. Therefore, the quality of the search can be quantified by a single proportion of correct building blocks in the population. This property simplifies the model to be derived.

In the model developed by Sastry, inherited fitness is taken as the mean of the parent fitness values. However, for analytical tractability, the mean of the parent fitnesses is taken as the mean building-block (BB) fitness. Fitness of a BB is the average fitness of all the individuals in the population that possess the schemata under consideration.

The model assumes that the actual fitness distribution, F, is Gaussian with mean $\mu_{f,t}$ and variance $\sigma_{f,t}^2$:

$$F = N(\mu_{f,t}, \sigma_{f,t}^2)$$

and that the distribution of fitness with inheritance, F', is also Gaussian with mean $\mu_{f',t}$ and variance $\sigma_{f',t}^2$:

$$F' = N(\mu_{f',t}, \sigma_{f',t}^2)$$

Therefore

$$\mu_{f',t} = \mu_{f,t}(1 - p_i) + \mu_{i,t} p_i \tag{6.10}$$

$$\sigma_{f',t}^2 = \sigma_{f,t}^2(1 - p_i) + \sigma_{i,t}^2 p_i \tag{6.11}$$

where $\mu_{i,t}$ and $\sigma^2_{i,t}$ are the fitness mean and variance, respectively, of individuals whose fitness is inherited (remember that p_i is the inheritance proportion). Since the inherited fitness, f_i, is equal to the average of BB fitness

$$f_i = \frac{1}{l}\sum_{j=1}^{l} \hat{f}(BB_j)$$

where, l is the string length, and $\hat{f}(BB_j)$ is the estimated BB fitness which can be written as

$$\hat{f}(BB_j) = f(BB_j) + (l-1)p_t$$

where, $f(BB_j)$ is the actual BB fitness, p_t is the proportion of correct BBs, and the term $(l-1)p_t$ incorporates the noise arising from other BBs. Using the above relation, f_i for uniformly scaled problems can be written as

$$f_i = \frac{f}{l} + (l-1)p_t.$$

The mean inherited fitness, $\mu_{i,t}$ is given by

$$\begin{aligned}
\mu_{i,t} &= \frac{1}{n}\sum_{j=1}^{n} f_{i,j} \\
&= \frac{1}{n}\sum_{j=1}^{n}\left[\frac{f_j}{l} + (l-1)p_t\right] \\
&= \frac{1}{l}\left[\frac{1}{n}\sum_{j=1}^{n} f_j\right] + \frac{1}{n}\sum_{j=1}^{n}(l-1)p_t \\
&= \frac{1}{l}\mu_{f,t} + (l-1)p_t \\
&= \frac{1}{l}lp_t + lp_t - p_t = lp_t = \mu_{f,t}
\end{aligned}$$

where n is the population size. Using the above relation in equation 6.10, we get:

$$\mu_{f',t} = \mu_{f,t}.$$

The inherited fitness variance, $\sigma^2_{i,t}$ can be derived as follows:

$$\begin{aligned}
\sigma^2_{i,t} &= \frac{1}{n}\sum_{j=1}^{n} f^2_{i,j} - (lp_t)^2 \\
&= \frac{1}{n}\sum_{j=1}^{n}\left[\frac{f_j}{l} + (l-1)p_t\right]^2 - (lp_t)^2 \\
&= \frac{1}{l^2}\frac{1}{n}\sum_{j=1}^{n} f_j^2 - (lp_t)^2 + \frac{2(l-1)p_t}{l}\frac{1}{n}\sum_{j=1}^{n} f_j + \frac{1}{n}\sum_{j=1}^{n}(l-1)^2 p_t^2 \\
&= \frac{1}{l^2}\left[\frac{1}{n}\sum_{j=1}^{n} f_j^2 - (lp_t)^2\right] - (l^2-1)p_t^2 + \frac{2(l-1)p_t}{l}(lp_t) + (l-1)^2 p_t^2 \\
&= \frac{1}{l^2}\sigma_{f,t} - (l^2-1)p_t^2 + 2(l-1)p_t^2 + (l-1)^2 p_t^2 \\
&= \frac{1}{l^2} lp_t(1-p_t) - (l^2-1)p_t^2 + (l^2-1)p_t^2 \\
&= \frac{p_t(1-p_t)}{l}
\end{aligned}$$

Using the above relation in equation 6.11, we get:

$$\begin{aligned}
\sigma^2_{f',t} &= (1-p_i)\sigma^2_{f,t} + p_i\frac{p_t(1-p_t)}{l} \\
&\approx (1-p_i)\sigma^2_{f,t}
\end{aligned}$$

Using the selection intensity equation[1] [90], we can write the expected average fitness with inheritance as:

$$\begin{aligned}
\mu_{f',t+1} &= \mu_{f',t} + I\sigma_{f',t} \\
&= \mu_{f',t} + I\sqrt{1-p_i}\sigma_{f,t}. \qquad (6.12)
\end{aligned}$$

Since both the actual fitness and inherited fitness distributions are normally distributed, the expected actual fitness value of F at generation $t+1$, given $\mu_{f',t+1}$ is:

$$E(F/\mu_{f',t+1}) = \mu_{f,t+1} = \mu_{f,t} + \frac{\sigma_{F,F'}}{\sigma^2_{f',t}}(\mu_{f',t+1} - \mu_{f',t})$$

[1] In Equation 6.12, I is the selection intensity and is defined as the expected increase in the average fitness of a population after selection is performed upon a population whose fitness is distributed according to a normal distribution.

It can be seen that the covariance, $\sigma_{F,F'}$ is $(1-p_i)\sigma_f^2 = \sigma_{f',t}^2$. Using this relation and Equation 6.12, we get:

$$\mu_{f,t+1} = \mu_{f,t} + I\sqrt{1-p_i}\sigma_{f,t} \tag{6.13}$$

Now substituting Equations 6.8 and 6.9 in 6.13:

$$\begin{aligned} lp_{t+1} &= lp_t + I\sqrt{1-p_i}\sqrt{lp_t(1-p_t)} \\ p_{t+1} &= p_t + I\sqrt{\frac{1-p_i}{l}}\sqrt{p_t(1-p_t)} \\ p_{t+1} - p_t &= I\sqrt{\frac{1-p_i}{l}}\sqrt{p_t(1-p_t)} \end{aligned} \tag{6.14}$$

Approximating the above equation as a differential equation yields:

$$\frac{dp}{dt} = \frac{I\sqrt{1-p_i}}{\sqrt{l}}\sqrt{p(1-p)}$$

Integrating the above equation and using the initial condition $p_0 = 0.5$ (since the initial population is generated with uniform distribution), we get:

$$p_t = \sin^2\left(\frac{\pi}{4} + \frac{I\sqrt{(1-p_i)}t}{2\sqrt{l}}\right) \tag{6.15}$$

An equation for convergence time can be derived by substituting $p_t = 1$, and inverting Equation 6.15:

$$t_{conv} = \frac{\pi}{2I}\sqrt{\frac{l}{(1-p_i)}}.$$

On the other hand, Sastry based his model for population sizing in the model previously developed by Miller [65]:[2]

$$n = -\frac{2^{k-1}log(\psi)\sqrt{\pi}}{d_{min}}\sqrt{\sigma_f^2},$$

where n is the population size, k is the BB length, ψ is the failure rate, d_{min} is the distance between the best BB and the second best BB, and $\sigma_{f'}^2$ is the variance of

[2]Miller developed a model for calculating the optimal population size required to ensure a good number of initial BBs supply and a good decision between competing BBs.

the fitness function. Not only $\sigma^2_{f'}$, but also d_{min} depends on p_i. For the OneMax problem, d_{min} was empirically determined to be:

$$d_{min} = (1-p_i^3)\sqrt{1-p_i}$$

The population sizing equation can now be written as:

$$n = -\frac{2^{k-1} log(\psi)\sqrt{\pi}}{(1-p_i^3)}\sqrt{\sigma^2_f},$$

Now, we proceed to determine the inheritance proportion such that the total number of function evaluations required is minimized. The total number of function evaluations is given by:

$$\begin{aligned} n_{f_e} &= n[(t_{conv}-1)(1-p_i)+1] \\ &= n[t_{conv}(1-p_i)+p_i] \end{aligned} \tag{6.16}$$

From the results previously obtained, we have:

$$t_{conv} = \frac{t_0}{\sqrt{1-p_i}},$$

$$n = \frac{n_0}{1-p_i^3}$$

where, $t_0 = t_{conv}(p_i = 0) = \pi\sqrt{l}/(2l)$, and $n_0 = n(p_i = 0) = -2^{k-1} log(\psi)\sqrt{\pi\sigma^2_f}/d_{min}$, are the convergence time and the population size when inheritance is not used. Using the above equations, an expression for the total number of function evaluations is given by:

$$n_{f_e} = \frac{n_0}{1-p_i^3}\left[t_0\sqrt{1-p_i}+p_i\right]$$

The optimal proportion inheritance is then given by solving:

$$\begin{aligned} \frac{\partial n_{f_e}}{\partial p_i} &= 0 \\ 3p_i^2\left[t_0(1-p_i)+p_i\sqrt{1-p_i}\right] + (1-p_i^3)\left[-0.5t_0+\sqrt{1-p_i}\right] &= 0 \end{aligned}$$

The above equation can be solved for two asymptotic cases: (1) the string lenght, $l = 0$, then $t_0 = 0$, and the optimal evaluates to $p_i^* = 0$, and (2) the string length is very long, then $t_0 \to \infty$. For this case the above equation reduces to:

$$3p_i^2(1-p_i) - \frac{1}{2}(1-p_i^3) = 0$$

$$p_i^2 - \frac{1}{5}p_i - \frac{1}{5} = 0$$

The above quadratic equation can be easily solved, and the optimal proportion for this case is:

$$p_i^* = 0.558$$

For other values of string length, Equation 6.17 cannot be solved analytically, and hence, it was solved numerically for different problems sizes. For moderate-to-large size problems the optimal proportion of inheritance values belong to the interval:

$$0.54 \leq p_i^* < 0.558$$

For the multi-objective case, Chen extended the OneMax problem. He defined the bicriteria OneMax problem as:

$$\text{Maximize} \begin{cases} f_1 = l - d(A, A_1) \\ f_2 = l - d(A, A_2) \end{cases}$$

where A is the string to be evaluated, A_1 and A_2 are two fixed strings with lenght l, and $d(A, A_i)$ is the Hamming distance function. If the fixed string A_i is an all-1s string, then the corresponding objective function f_i will be that of the OneMax problem. The number of Pareto-optimal solutions, Q, in the bicriteria OneMax problem can be calculated by

$$Q = 2^{d(A_1, A_2)}$$

In the analysis developed by Chen, A_1 is an all-1s string, and A_2 is an all-1s string except for the last b bits of A_2 which are 0s. Therefore, $d(A_1, A_2) = b$.

Chen derived the convergence model for the bicriteria OneMax problem by extending Equation 6.13.[3] Based on the concept of fitness sharing, Chen assumes that the population is divided into several subpopulations (niches), and each niche optimizes its own separate OneMax problem. Therefore, the optimizing process for the bicriteria OneMax problem can be regarded as optimizing several OneMax problems simultaneously. Since niches are from the same population, each niche will receive external noise from other niches. For each niche, the convergence

[3] It is worth noting that, in [12, 11], Chen indicates that in the case of the bicriteria One Max problem, he inherits the values of the two objective functions instead of the fitness values. However, the model developed by Chen is based on the same premises of the model developed by Sastry.

model can be expressed as:

$$\mu_{f,t+1} = \mu_{f,t} + I\sqrt{1-p_i}\frac{\sigma_f^2}{\sqrt{\sigma_f^2+\sigma_N^2}}$$

where σ_N^2 is the noise variance from other niches. Let M be the number of niches in the population, and

$$\rho_e = \frac{\sqrt{\sigma_f^2+\sigma_N^2}}{\sigma_f^2}.$$

The noise variance from other niches can be approximated by $(M-1)p_t(1-p_t)$. In this way, Chen generalizes Equations 6.14 and 6.15:[4]

$$p_{t+1} - p_t = \frac{I}{\rho_e}\sqrt{\frac{1-p_i}{l}}\sqrt{p_t(1-p_t)}$$

$$p_t = \sin^2\left(\frac{\pi}{4} + \frac{I\sqrt{(1-p_i)}t}{2\rho_e\sqrt{l}}\right)$$

From which, Chen obtains:[5]

$$\begin{aligned} t_{conv} &= \frac{\pi}{2I}\sqrt{\frac{l}{(1-p_i)}}\rho_e \\ &= \frac{\pi}{2I}\sqrt{\frac{l}{(1-p_i)}}\sqrt{1+\frac{M-1}{l}} \end{aligned} \tag{6.17}$$

On the other hand, assuming the population is divided into M niches, and each niche optimizes its own separate OneMax problem, Chen extended the model derived by Sastry for predicting the optimal population size:

$$n = -\frac{2^{k-1}log(\psi)M\sqrt{\pi}}{(1-p_i^3)}\sqrt{\sigma_f^2+\sigma_N^2}, \tag{6.18}$$

[4] In this case, we don't provide details since we were not able to obtain them, from the analysis provided by Chen in [12, 11]

[5] It is important to note that, as in the case of the single objective version of the One Max problem, Chen considers that convergence has been reached when $p_t = 1$ (that is, when the whole population is composed of correct BBs). However, this assumption is not justified by Chen in the case of the multi-objective version of the One Max problem.

Finally, similar to the process developed by Sastry, from Equations 6.16, 6.17 and 6.18, Chen obtains:

$$p_i^* = \sqrt[3]{1 - \frac{\kappa}{n}}$$

where $\kappa = -2^{k-1} log(\psi) M \sqrt{\pi(\sigma_f^2 + \sigma_N^2)}$.

6.3 Fitness Inheritance for a real-coded MOPSO

As we could see in the previous section, the existing studies about fitness inheritance are based on models in which binary representation is used. Also, the models previously presented analyze the properties of fitness inheritance under the assumption of a very particular problem (OneMax). In this section, we present a simple analysis (*discussion*) of the impact of a fitness inheritance technique applied on real-valued vectors, for the particular case of the MOPSO approach described in Chapter 4. As we mentioned previously, with this analysis we attempt to explore the possible reasons of the effect that fitness inheritance has on the quality of the results provided by our MOPSO approach (as we could see in Chapter 5).

In Chapter 5, we described fifteen different fitness inheritance techniques, which were incorporated into the MOPSO approach previously described. In all cases, when inheritance is applied, a particle inherits its new objective function values (after updating its position in search space) from the corresponding values of its parents, by means of a specific mechanism (different for each inheritance technique). According to the results provided by the techniques proposed, we found that the best of them were based on two main concepts:

- Linear combination of the objective values of the parents.
- Formula to update positions (flight) in the objective space.

As we could see in Section 6.1, when applying the formula of PSO for updating positions of particles, in the search space, we obtain convergence to a linear combination of the corresponding *pbest* and *gbest* positions. In this way, we decided to analyze one of the fitness inheritance techniques proposed in Chapter 5, called FI2, which calculates a linear combination of the *pbest* and *gbest* positions, in objective function space. By analyzing technique FI2, we aim to explore

some of the properties of the fitness inheritance techniques proposed in Chapter 5, considering technique FI2 as the general case.

First of all, it is important to note that, in the implementations performed of the fitness inheritance techniques previously proposed, we never let a particle with inherited objective values, enter into the available set of leaders. In this way, all available leaders have true objective values. Thus, we can conclude that a particle which inherits its objective values does not affect directly the trajectory of the rest of the particles of the swarm. That is, a particle which inherits its function values only affects directly its own trajectory.[6] This property allows us to analyze the effects of inheritance on the trajectory isolated of a single particle.

We remember here the equations for updating the position of the particles in PSO:

$$\mathbf{x}_i(t) = \mathbf{x}_i(t-1) + \mathbf{v}_i(t) \tag{6.19}$$

$$\mathbf{v}_i(t) = W\mathbf{v}_i(t-1) + C_1 r_1(pbest_i - \mathbf{x}_i(t-1)) + C_2 r_2(leader_i - \mathbf{x}_i(t-1)) \tag{6.20}$$

where *pbest* represents the personal best position of particle *x* and *leader* represents the *gbest* solution selected by particle *x* from the available set of leaders. As we mentioned before, the *leader* particle will always have true objective function values. In this way, it remains to explore how the *pbest* particle is affected when particle *x* inherits its objective function values.

As we described in Chapter 4, our MOPSO approach updates the *pbest* position of a particle *x* when its new position:

1. dominates the current *pbest* position.

2. is incomparable to the current *pbest* position.

In this way, the only case in which the *pbest* position is not updated with the new position of the particle is when the new position of the particle is dominated by the current *pbest* position. We can see all possibles cases in Figure 6.1. From Figure 6.1, we can conclude that in a given time step t, the *pbest* position of a particle is either the current position of the particle itself or a position that dominates the current position of the particle.

On the other hand, when a particle selects a leader in order to update its position in the search space, such leader (by definition) either dominates or is incomparable to the particle. For this reason, given the previous discussion about the

[6] It is clear that a particle which inherits its objective function values affects *indirectly* the trajectory of the other particles of the swarm, but this effect will be discussed later.

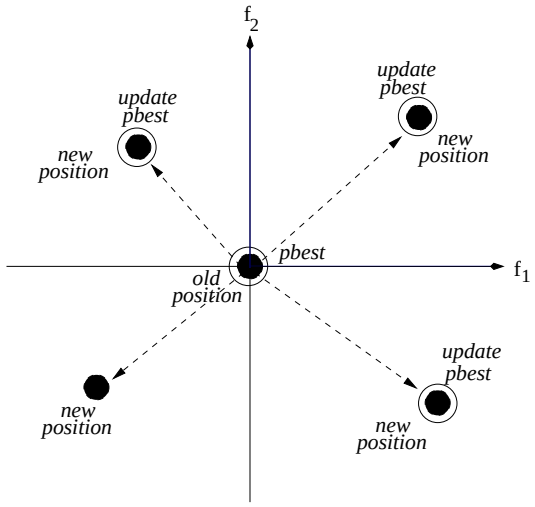


Figure 6.1: Possible directions in which a particle can move in the objective space (assuming a bi-objective maximization case). Only when the new position is dominated by the previous one, the *pbest* position is not updated.

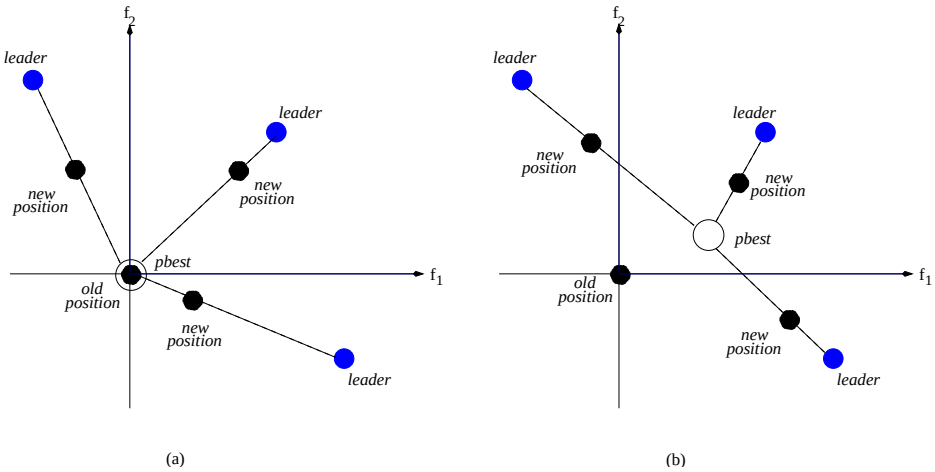


Figure 6.2: Possible positions of a linear combination of the *pbest* position and the leader: (a) when the *pbest* position is the current position of the particle and (b) when the *pbest* position dominates the current position of the particle. The resulting (new) position always dominates or is incomparable to the (old) position of the particle.

location of the *pbest* position, any linear combination of the *pbest* position and the leader will either dominate or be incomparable to the current position of the particle. In fact, such linear combination will either dominate or be incomparable to the current *pbest* position of the particle. Figure 6.2 shows the possible cases, in a bi-objective maximization example, for the position of the linear combination calculated for inheriting objective function values.

As a conclusion, we have that the new position (in objective function space) of a particle which inherits its objective function values will either dominate or be incomparable to the *pbest* position of the particle. In this way, based on the mechanism used by our MOPSO approach for updating the *pbest* position, we can conclude that when a particle inherits its objective function values, the corresponding *pbest* position is always updated.

Thus, we have that when a particle with inherited objective function values updates its position in the search space, the term related to the *pbest* position in the velocity formula vanishes:

$$C_1 r_1 (pbest_i - \mathbf{x}_i(t-1)) = 0,$$

and the obtained velocity vector is:

$$\mathbf{v}_i(t) = W\mathbf{v}_i(t-1) + C_2 r_2 (leader_i - \mathbf{x}_i(t-1)).$$

As we can see, the first (immediate) consequence of inheritance in the trajectory of a single particle is the elimination of the direction to the *pbest* position in the velocity vector. In this way, inheritance eliminates from the velocity vector, the term that allows the particle to return to the previous position, in the case that the new position is not better. Thus, we can conclude that the impact of inheritance is negative on the velocity vector (and in the trajectory of the particle) only in the case that the last movement takes the particle to a dominated position.

Other consequences of inheritance in the trajectory of a particle are important. In fact, the *pbest* position is the one that always suffers the impact of inheritance. For example, once a particle has inherited its objective function values and updated its *pbest* position (without true knowledge of its quality), the process of updating the *pbest* position in subsequent steps becomes difficult because the *pbest* position has no true objective function values. In this way, every time we compare the objective function values of the *pbest* position against new objective function values of the particle (either true or inherited), the winner is not necessarily the best. In this point, it is important to note that, in the particular case of our MOPSO approach, a particle with true objective function values is allowed to (try

to) enter into the set of available leaders only if in the last movement, the particle was able to improve its *pbest* position. Thus, inheritance might prevent nondominated particles with true objective function values of entering into the available set of leaders due to the inherited objective function values assigned to the *pbest* position in previous steps.

On the other hand, the inherited objective function values of the *pbest* position of the particle also affect the new objective function position of the particle if inheritance is applied again in subsequent steps. Since the new inherited objective function values are calculated using a linear combination of the *pbest* position and the leader, the obtained values are based on values that are not necessarily true. We may call this effect *error transmission*.

In this way, we can conclude that fitness inheritance affects directly the trajectory of a particle through the *pbest* solution, in two main forms:

E1 Eliminating from the velocity vector the term that allows the particle to turn back to the previous position, in the case that the new one is not better (when inheritance is applied for the first time).

E2 Preventing the *pbest* position of being correctly updated, due to the inherited objective values assigned to it, in previous steps.

In the second case, the *pbest* position introduces some kind of *noise* in the trajectory of a particle, since it introduces, into the velocity vector, a direction to a particle of uncertain quality. This effect may be considered as a way of introducing some diversity into the swarm.

Thus, fitness inheritance affects the trajectory of a particle through the corresponding *pbest* position. However, it is convenient to remember that the ANOVA described in Chapter 4 indicated that the value of the parameter C_1 in Equation 6.20 is not important for the behavior of the algorithm. That is, the *pbest* position does not have an important impact in the trajectory of particles neither on the performance of the algorithm. For this reason, we may argue that this effect of fitness inheritance on the trajectory of particles is not the most important factor to consider, or at least not by itself alone.

On the other hand, there are two main ways by which fitness inheritance affects *indirectly* the trajectory of particles and, in general, the performance of the algorithm:

E3 A particle which has inherited its objective function values is not allowed to (try to) enter into the set of available leaders.

E4 A particle with true objective function values may not be able to improve its corresponding *pbest* position (due to the objective function values inherited in previous steps) and, consequently, it may not be able to (try to) enter into the set of available leaders.

The first consequence of these two effects of fitness inheritance is the corresponding size of the set of leaders. However, the most important consequence is the loss of information. As we previously discussed, inheritance affects immediately the information stored in the *pbest* position of a particle. Thus, the selection of a good leader becomes crucial for the performance of the algorithm. In this way, we can conclude that inheritance decreases the number of solutions reported by the algorithm but also the quality of the final set of solutions, since the set of leaders (which guide the search) is not correctly updated throughout the evolutionary process.

As we could see, fitness inheritance might have negative effects on the performance of the MOPSO algorithm proposed in two main cases:

- When a particle updates its *pbest* position incorrectly, as a result of inheritance. That is, when the *pbest* position of a particle is updated with a dominated position.
- When a particle is not allowed to (try to) enter into the available set of leaders. This case is possible even when the particle has true objective function values, as we previously discussed.

In this way, in order to explore how important is the effect of inheritance in the performance of the algorithm, we would like to answer the following questions:

- How often a particle moves to a dominated position (by the *pbest* position)? That is, how often a particle does not improve its *pbest* position?
- How much is the size of the set of leaders affected by inheritance?

The second question is related to the first one, since the answer to the first question would give us an idea of how often a particle intends to enter into the set of leaders. In fact, the value of the inheritance proportion (the probability by which a particle inherits its objective values) gives us a very clear idea of how many particles will not be able to enter into the set of leaders. Thus, it would remain to explore how often a particle with true objective values is not able to improve its *pbest* position. However, this last case seems difficult to study.

We performed some experiments using our MOPSO approach and the inheritance technique FI2, in order to obtain an empirical answer to the questions previously mentioned. We ran our algorithm 30 times with and without inheritance. The values of inheritance proportion used were: p_i= 0.0 (without inheritance), 0.1, 0.3, 0.6 (we maintained the value of p_i fixed throughout the run). The parameters adopted were a swarm size of 100 particles and 100 generations. The test functions used were ZDT1, ZDT2, ZDT3 and ZDT4. For each function and each value of inheritance proportion we calculated two values:

- Average number of particles that improved its corresponding *pbest* position, at each generation.
- Average number of leaders at each generation.[7]

Figures 6.3, 6.4, 6.5 and 6.6, show the results obtained, for the average number of particles that improved its corresponding *pbest* position, at each generation. For each function, we present two plots. The first plot shows the results corresponding to the case of our algorithm without inheritance and the second plot shows the results of our algorithm with all the different inheritance proportion values used. Also, in all cases, we plot a constant function with value 100, that would represent the case of all particles improving its *pbest* position at each generation.

As we can see in the results of our algorithm without inheritance, the average number of particles that improve its *pbest* position at each generation is very high. This average has values over the 80% of particles, for all the cases. In fact, for functions ZDT1 and ZDT2 almost 100% of particles improve its *pbest* position at each generation. On the other hand, for function ZDT4, the corresponding average is over the 90% of particles. With these results, we can conclude that a particle almost never moves to a position dominated by its current *pbest* position (which is the only case in which the *pbest* position is not updated). In this way, we may conclude that the effect **E1** of inheritance when forcing a particle to replace its current *pbest* position does not significantly affect in a negative way, the performance of the algorithm.

We can see that the results of our algorithm with inheritance are, in general, very similar to the corresponding results without inheritance. In some cases, we can see that the use of inheritance decreases marginally the average of particles

[7] In this case, we record the total number of leaders obtained after updating the current set of leaders, but before reducing it to the maximum size allowed. Since the swarm size and the maximum number of leaders used were of 100 particles, the total number of leaders lies between 100 and 200, at each generation.

that improve its *pbest* position, especially at the beginning of the process. This effect is more noticeable in functions ZDT1 and ZDT3 (see Figures 6.3 and 6.5). We conclude that this behavior is due to the effect **E2** of inheritance, when the process of updating the *pbest* position is affected by the objective function values inherited in previous generations. On the other hand, we can see that the use of inheritance also increases the same average value by the end of the process. This behavior is due to the effect **E1** of inheritance (previously discussed) and is more noticeable in function ZDT3 (see Figure 6.5). In general, we can observe that the behavior of our algorithm, produced by the use of inheritance, is more stressed when the value of inheritance proportion increases (see, for example, Figure 6.5).

On the other hand, Figures 6.7, 6.8, 6.9 and 6.10, show the results obtained, for the average number of leaders at each generation. As we can see, the number of leaders decreases as the value of inheritance proportion increases, for all cases. However, the impact is not as noticeable as we would have expected. In the worst case, when the biggest value of inheritance proportion is used ($pi = 0.6$), the average number of leaders decreases by 30% with respect to the corresponding average when inheritance is not used. Also, we can observe in Figure 6.8 that, in some cases, the average of the number of leaders at each generation grows when $p_i = 0.1$. This may be the reason for the improvement on the quality of the results shown in Chapter 5, when $p_i = 0.1$.

Thus, we have seen that the most important effect of fitness inheritance in the evolutionary process of our MOPSO approach is the decrement of the size of the set of available leaders. In this way, we may conclude that this behavior is the one that has the most important impact on the quality of the final set of solutions (as it was observed in the experiments presented in Chapter 5), since such set of leaders guides the search. On the other hand, although we could see that the impact of inheritance when forcing a particle to update its *pbest* position is not very important, it remains to study the effects that the *error transmission* discussed before has on the quality of the results provided by the algorithm.

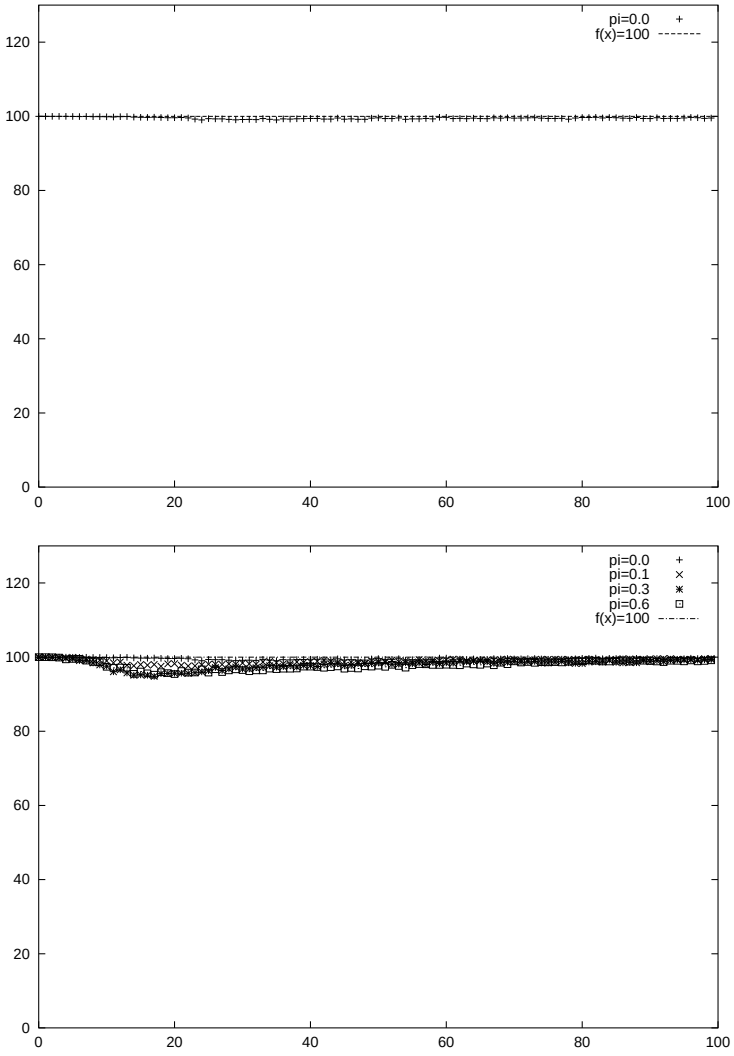


Figure 6.3: Average number of particles that improved its corresponding *pbest* position, at each generation, for test function ZDT1.

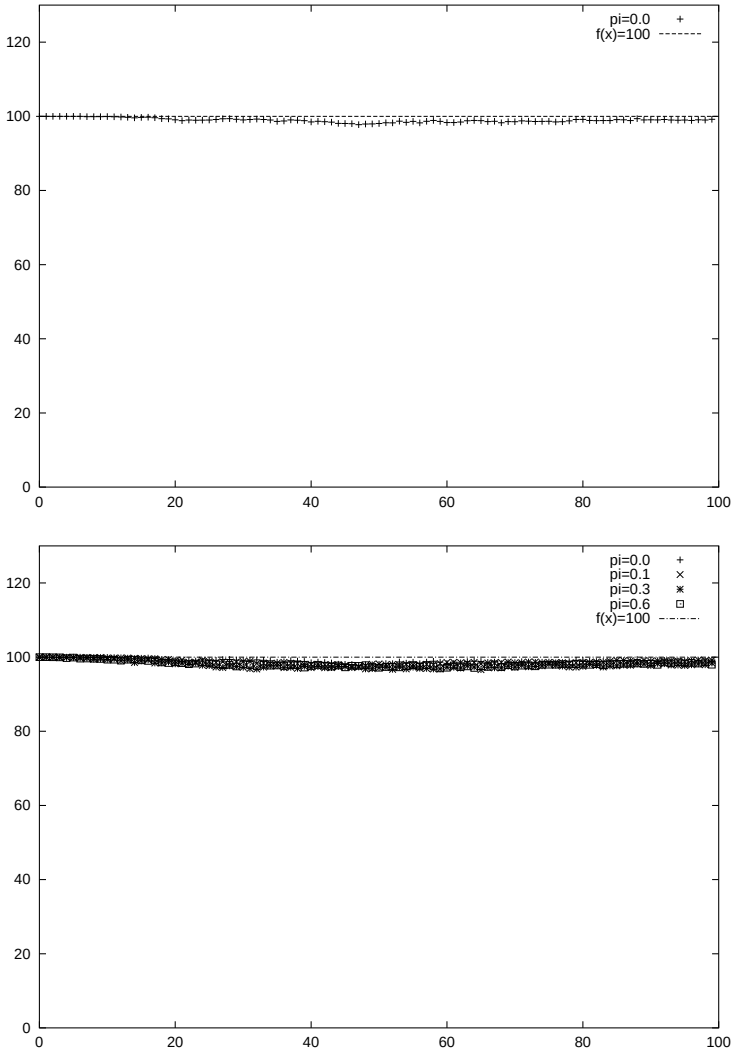


Figure 6.4: Average number of particles that improved its corresponding *pbest* position, at each generation, for test function ZDT2.

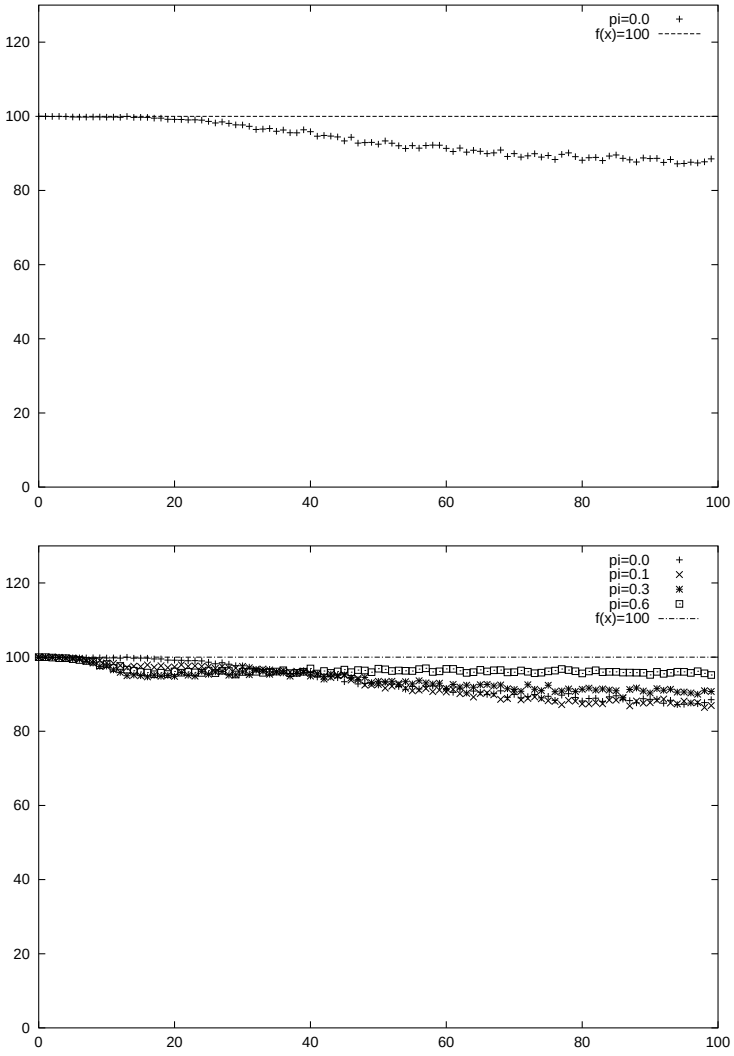


Figure 6.5: Average number of particles that improved its corresponding *pbest* position, at each generation, for test function ZDT3.

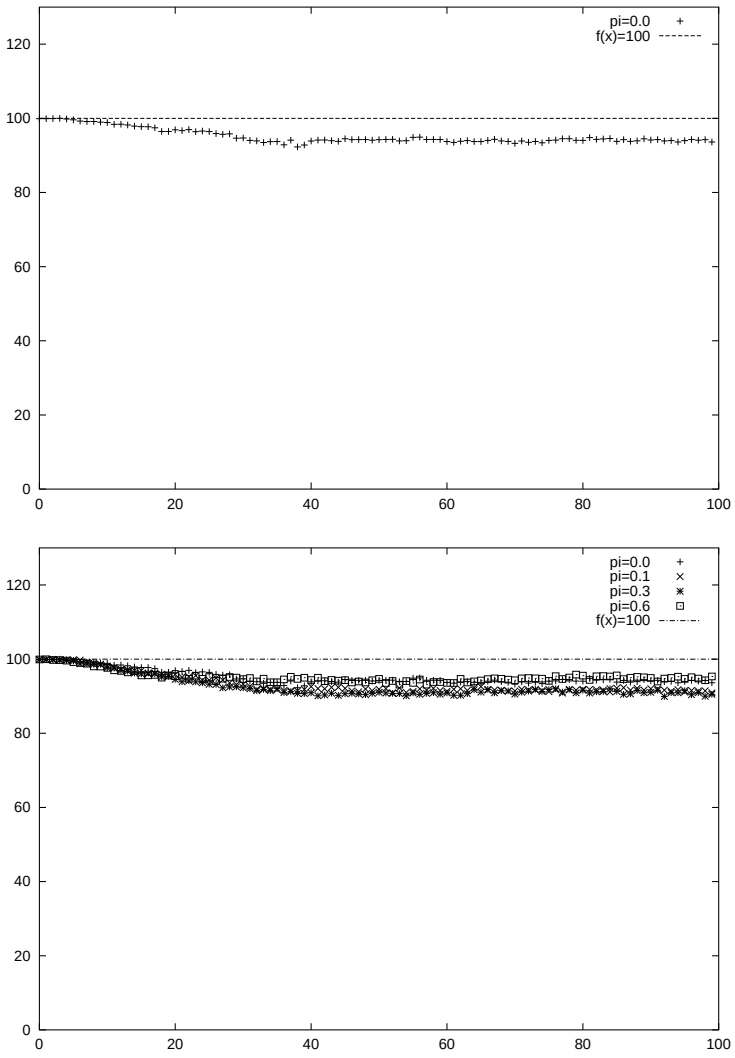


Figure 6.6: Average number of particles that improved its corresponding *pbest* position, at each generation, for test function ZDT4.

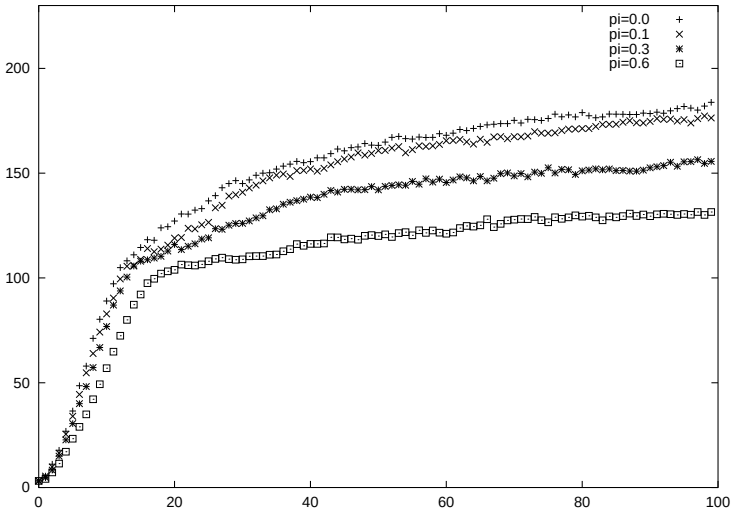


Figure 6.7: Average number of leaders at each generation, for function ZDT1.

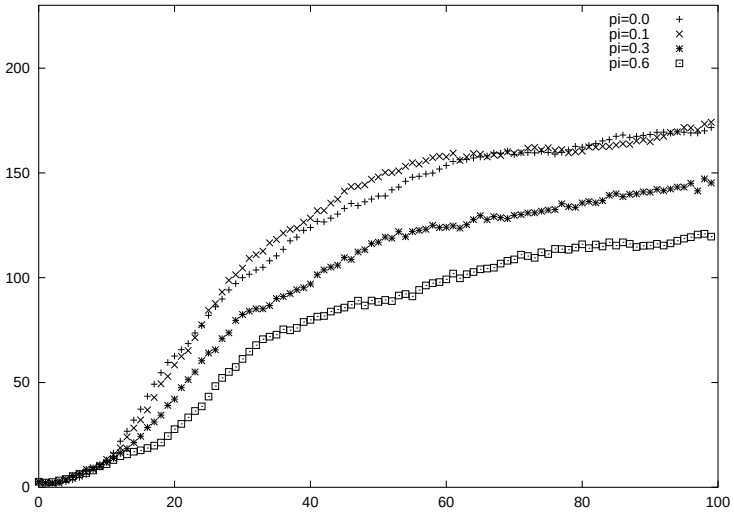


Figure 6.8: Average number of leaders at each generation, for function ZDT2.

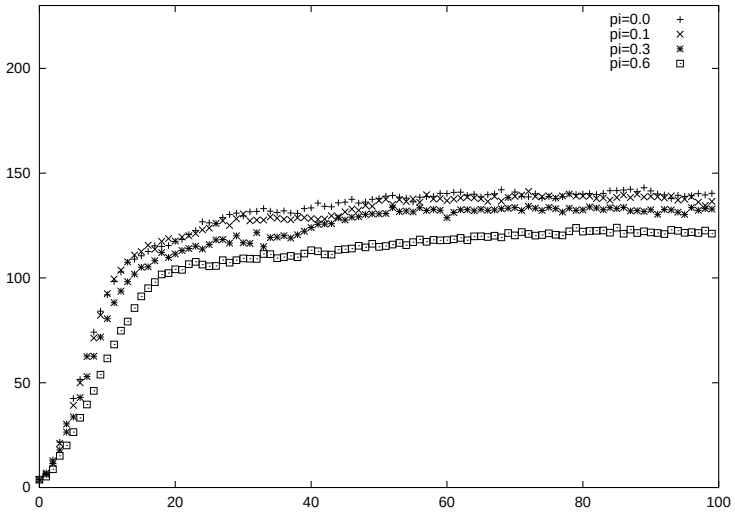


Figure 6.9: Average number of leaders at each generation, for function ZDT3.

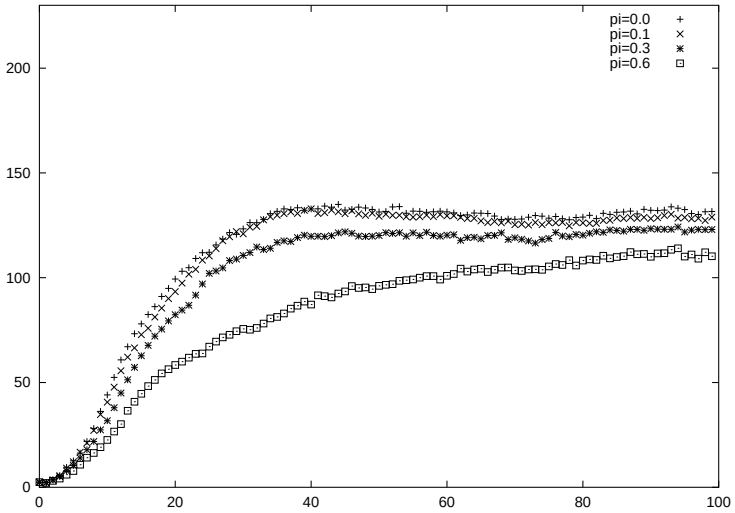


Figure 6.10: Average number of leaders at each generation, for function ZDT4.

Chapter 7

Final Remarks

Given the population-based nature of EAs, their application to real-world problems whose objective function cost is computationally expensive is limited.

With the aim of designing new mechanisms able to reduce computational cost (in terms of the number of function evaluations performed), in the first part of this work we presented a coevolutionary multi-objective algorithm whose main idea was to detect the most "promising" sub-regions of the search space and focus the search on them. With this aim, the algorithm applied a clustering technique on the set of decision variables of the Pareto front known so far. Thus, by obtaining information along the evolutionary process, the proposed approach was supposed to ignore the useless sub-regions of the search space and to reduce the computational cost involved in the process of convergence to the optimal solutions.

After doing experiments with test functions with high-dimensional decision spaces, we could observe that our coevolutionary algorithm was not able to converge to the true Pareto fronts of the problems. For this reason, we designed a new PSO-based multi-objective algorithm to be used as a search engine (replacing the genetic algorithm originally used).

The proposed PSO-based approach was found to be highly competitive. However, once incorporated into our coevolutionary approach, we didn't obtain the expected results. In fact, the obtained results indicated that the coevolutionary approach was not able to match the results of the MOPSO approach alone, while performing the same number of function evaluations. Thus, we concluded that our coevolutionary scheme had some limitations when solving the problems with high dimensional decision spaces used in our study. For this reason, we decided to study different mechanisms for reducing computational cost.

In this way, we proposed the use of fitness inheritance in order to reduce the

computational cost of the PSO-based multi-objective approach provided before. Since the results obtained by our first attempt were very promising, we studied several different ways of incorporating such enhancement technique and also experimented with some simple fitness approximation schemes. Afterwards, since both fitness inheritance and approximation techniques provide a reduction in the computational cost that is completely determined by the value of inheritance (or approximation) proportion p_i, we proposed several schemes that set the value of this parameter in a dynamical way throughout the evolutionary process. The results provided by the proposed approaches showed that fitness inheritance enables our MOPSO approach to obtain better results than two algorithms representative of the state-of-the-art, while performing a smaller number of function evaluations.

Finally, we presented some theoretical studies that allowed us to analyze the convergence properties of the multi-objective particle swarm approach and the effects of the fitness inheritance techniques proposed.

7.1 Conclusions

- Coevolutionary schemes that partition the search space are not suitable for solving the test problems with high-dimensional decision spaces used in our study. In particular, the proposed coevolutionary scheme does not seem to be useful for the set of test functions adopted, which consists of very specific continuous real-valued functions (that is precisely the type of problem in which we are interested). However, it remains open the posibility of testing the usefulness of such scheme when using it for solving different types of problems.

- The particle swarm optimization strategy is very effective for solving multi-objective problems. In particular, it was successful when solving the test functions with high-dimensional decision spaces adopted in our study.

- Although the proposed MOPSO approach has some difficulties solving the test problems with nonuniform density of solutions in the search space, from the set of test functions adopted, it was successful for solving multimodal problems. It is worth noting that our approach was the only PSO-based algorithm (out of four adopted in our study) able to solve a test problem with 21^9 local Pareto optimal solutions in decision variable space (a total of 100 distinct local Pareto fronts in objective function space).

- Leader selection is a key component when designing PSO-based multi-objective approaches. The ANOVA performed, let us realize that the value of the parameter related to leader selection in our approach has a very important impact on the performance of our algorithm, for the test functions adopted, especially in those with nonuniform density of solutions in the search space.

- From the statistical analysis performed, we were able to propose on-line adaptation mechanisms for the parameters of our MOPSO approach, that provided good results for the test functions adopted. Nevertheless, the ANOVA also provided some guidelines to set the values of the parameters of our approach, if we have some *a priori* knowledge about the problem to be solved.

- Although for the test function used, fitness inheritance affects the quality of the provided Pareto fronts as we increase the number of true evaluations saved, we were able to save a 32% of the total number of evaluations without significantly deteriorating the quality of the results. In fact, although the quantitative quality of the Pareto fronts provided by the approaches that save 65% and 78% of evaluations is more affected, the corresponding plots show that such approaches are able to generate good approximations of the true Pareto front.

- When comparing the results provided by the proposed fitness inheritance approach with respect to two of the most representative algorithms of the state-of-the-art (the NSGA-II and the SPEA2), we conclude that the proposed mechanism enables the MOPSO approach previously described, to obtain better results while performing a lower number of function evaluations, for the test functions adopted.

- From our results obtained in the test functions adopted, we may conclude that the effect of fitness inheritance on the quality of the results is less noticeable in test functions with decision spaces of high dimensionality.

- Also, it seems that fitness inheritance has a greater impact on the quality of the obtained results, as the number of objectives increases.

- From the results found in the literature, we were able to conclude the convergence of our MOPSO approach to the true Pareto front.

- The most important effect of fitness inheritance in the evolutionary process of our MOPSO approach is the decrement of the size of the set of leaders and the *error transmission*.

7.2 Future Work

Part of our future work includes the following:

- To study different coevolutionary algorithms able to reduce computational cost.
- To apply the proposed MOPSO approach and the fitness inheritance techniques proposed to real-world problems.
- To test the fitness inheritance techniques proposed on different evolutionary algorithms. Also, to test fitness inheritance techniques on different test functions in order to explore the corresponding performance when higher dimensional problems are adopted.
- Since we applied the concept of fitness inheritance is such a way that we reduce the computational cost and we explored how the quality was affected, it remains to apply inheritance by maintaining fixed the number of evaluations and exploring the changes on quality when we allow the evolutionary process to run during a larger number of generations.
- To study the so-called *error transmission*, in order to propose mechanisms by which we are able to maintain the quality of the solutions throughout the evolutionary process, despite the use of fitness inheritance.
- To explore different mechanisms to reduce computational cost, like surrogate models [109].

Appendix A

Results provided by the MOPSO approach proposed

Test Function ZDT1							
		OMOPSO	NSGA-II	SPEA2	MOPSO	sMOPSO	cMOPSO
SCC	best	84	38	47	0	93	37
	median	74	20	26	0	58	7
	worst	22	9	15	0	23	1
	average	**71**	21	27	0	59	8
	std. dev.	13.6	7.5	8.1	0	24.2	7.9
IGD	best	0.0009	0.0008	0.0006	0.0240	0.0031	0.0016
	median	0.0010	0.0008	0.0007	0.0276	0.0260	0.0029
	worst	0.0011	0.0011	0.0008	0.0385	0.0448	0.0041
	average	0.0010	0.0009	**0.0007**	0.0286	0.0269	0.0030
	std. dev.	0.00005	0.00009	0.00006	0.0040	0.0095	0.0007

Table A.1: Comparison of results between our approach (denoted by OMOPSO), NSGA-II [27], SPEA2 [118], MOPSO [17], sMOPSO [68] and cMOPSO [103], for test function ZDT1 with respect to the unary measures.

Test Function ZDT1 - Two Set Coverage Measure SC						
SC(X,	OMOPSO)	NSGA-II)	SPEA2)	MOPSO)	sMOPSO)	cMOPSO)
OMOPSO	0.00	**0.76**	**0.68**	**0.96**	0.08	**0.95**
NSGA-II	0.00	0.00	0.22	1.00	0.01	0.99
SPEA2	0.00	0.49	0.00	1.00	0.01	1.00
MOPSO	0.00	0.00	0.00	0.00	0.00	0.00
sMOPSO	**0.35**	0.69	0.66	0.99	0.00	0.90
cMOPSO	0.00	0.01	0.00	1.00	0.00	0.00
Test Function ZDT1 - Two Set Hypervolume Measure HV						
HV(X,	OMOPSO)	NSGA-II)	SPEA2)	MOPSO)	sMOPSO)	cMOPSO)
OMOPSO	0.000000	-0.003267	-0.002367	-0.325765	0.006035	-0.021381
NSGA-II	0.001942	0.000000	-0.000037	-0.322274	0.007647	-0.016607
SPEA2	0.002345	-0.000534	0.000000	-0.322771	0.007733	-0.017104
MOPSO	0.001718	0.000000	0.000000	0.000000	0.000000	0.000000
sMOPSO	0.000356	-0.003241	-0.002658	-0.333162	0.000000	-0.020032
cMOPSO	0.000435	0.000000	0.000000	-0.305667	0.007463	0.000000

Table A.2: Comparison of results using the binary measures for test function ZDT1. Our algorithm is denoted by OMOPSO.

Test Function ZDT2							
		OMOPSO	NSGA-II	SPEA2	MOPSO	sMOPSO	cMOPSO
SCC	best	100	30	34	0	1	94
	median	91	0	0	0	1	0
	worst	60	0	0	0	1	0
	average	**89**	6	7	0	1	29
	std. dev.	10	9.8	10.4	0	0	38.9
IGD	best	0.0006	0.0008	0.0007	0.0271	0.0723	0.0030
	median	0.0007	0.0724	0.0723	0.1098	0.0723	0.0723
	worst	0.0008	0.0737	0.0736	0.3525	0.0723	0.0852
	average	**0.0007**	0.0512	0.0404	0.1561	0.0723	0.0680
	std. dev.	0.00005	0.0337	0.0367	0.0952	0.0000	0.0152

Table A.3: Comparison of results between our approach (denoted by OMOPSO), NSGA-II [27], SPEA2 [118], MOPSO [17], sMOPSO [68] and cMOPSO [103], for test function ZDT2 with respect to the unary measures.

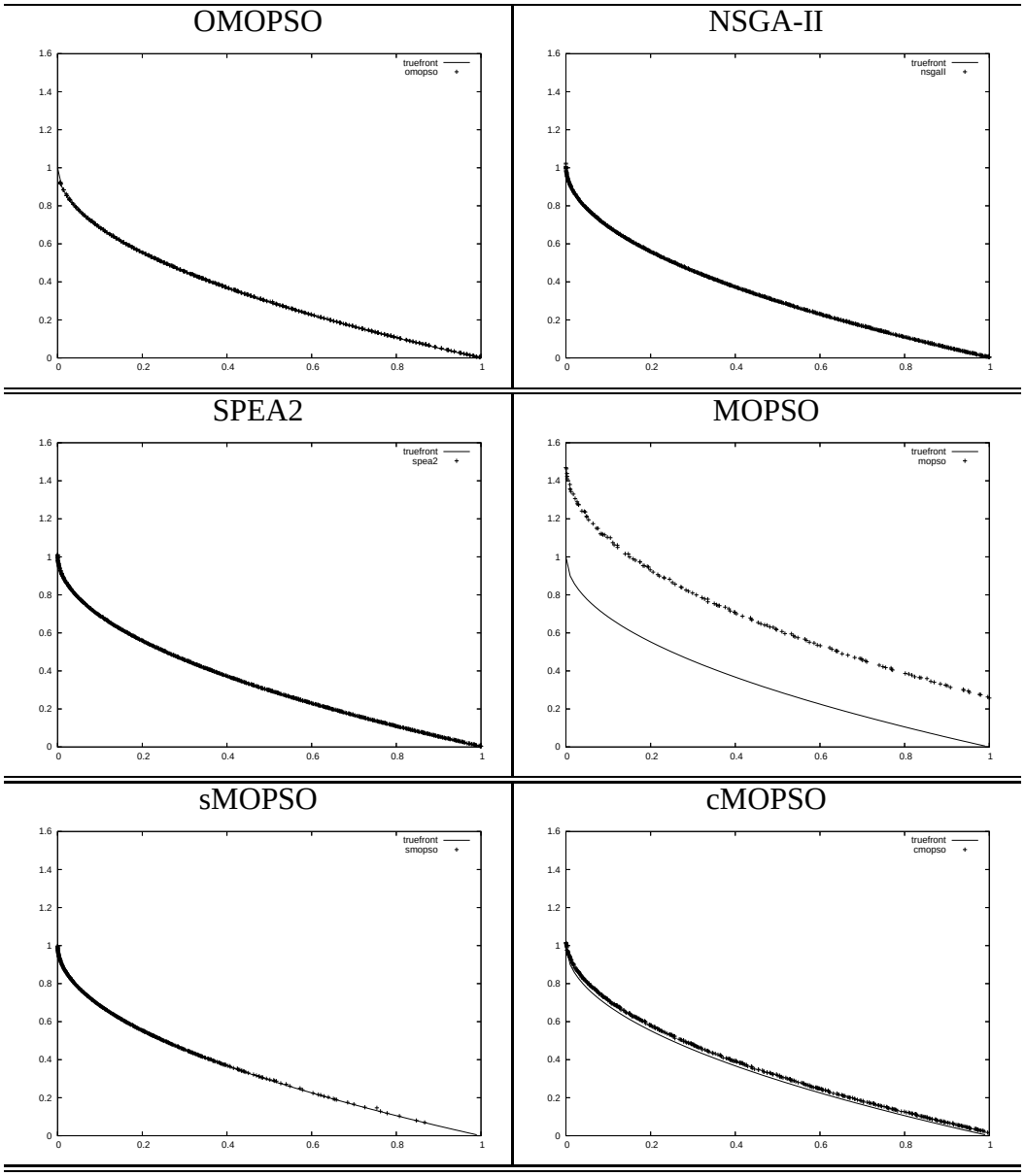


Figure A.1: Pareto fronts obtained by all the approaches for test function ZDT1. Our algorithm is denoted by OMOPSO and, in this case, it used ε=0.0075.

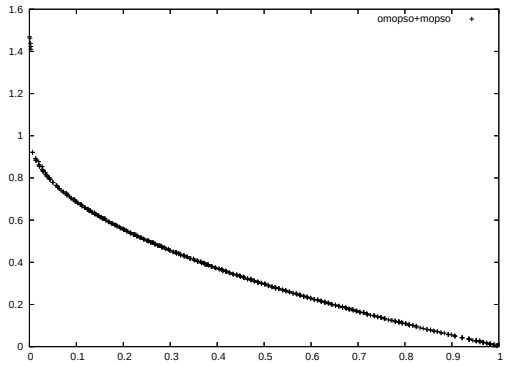


Figure A.2: Pareto front obtained from the union of the fronts obtained by OMOPSO and MOPSO, in the first test function.

Test Function ZDT2 - Two Set Coverage Measure SC						
SC(X,	OMOPSO)	NSGA-II)	SPEA2)	MOPSO)	sMOPSO)	cMOPSO)
OMOPSO	0.00	**0.92**	**0.93**	**1.00**	0.00	**0.21**
NSGA-II	0.00	0.00	0.34	1.00	0.00	0.21
SPEA2	0.00	0.21	0.00	1.00	0.00	0.21
MOPSO	0.00	0.00	0.00	0.00	0.00	0.00
sMOPSO	0.00	0.01	0.01	0.44	0.00	0.00
cMOPSO	0.00	0.02	0.02	0.99	0.00	0.00
Test Function ZDT2 - Two Set Hypervolume Measure HV						
HV(X,	OMOPSO)	NSGA-II)	SPEA2)	MOPSO)	sMOPSO)	cMOPSO)
OMOPSO	0.000000	-0.006493	-0.006565	**-0.343317**	**-0.665454***	-0.037835
NSGA-II	0.001015	0.000000	-0.000493	-0.336593	-0.672178*	-0.031559
SPEA2	0.001138	0.000486	0.000000	-0.335614	-0.673157*	-0.030560
MOPSO	**0.000000**	0.000000	0.000000	0.000000	-0.897843*	0.000000
sMOPSO	**0.000000**	0.000000	0.000000	-0.110928	0.000000	0.000000
cMOPSO	0.000231	-0.000217	-0.000197	-0.305251	-0.703520*	0.000000

Table A.4: Comparison of results using the binary measures for test function ZDT2. Our algorithm is denoted by OMOPSO.

Test Function ZDT3							
		OMOPSO	NSGA-II	SPEA2	MOPSO	sMOPSO	cMOPSO
SCC	best	90	56	50	0	89	0
	median	72	42	39	0	15	0
	worst	18	33	25	0	0	0
	average	**68**	44	39	0	26	0
	std. dev.	18.2	6.8	6.0	0	25.4	0
IGD	best	0.0008	0.0008	0.0007	0.0281	0.0023	0.0028
	median	0.0008	0.0009	0.0009	0.0316	0.0249	0.0054
	worst	0.0021	0.0104	0.0106	0.0447	0.0374	0.0096
	average	**0.0009**	0.0013	0.0018	0.0334	0.0245	0.0062
	std. dev.	0.0003	0.0021	0.0030	0.0048	0.0095	0.0020

Table A.5: Comparison of results between our approach (denoted by OMOPSO), NSGA-II [27], SPEA2 [118], MOPSO [17], sMOPSO [68] and cMOPSO [103], for test function ZDT3 with respect to the unary measures.

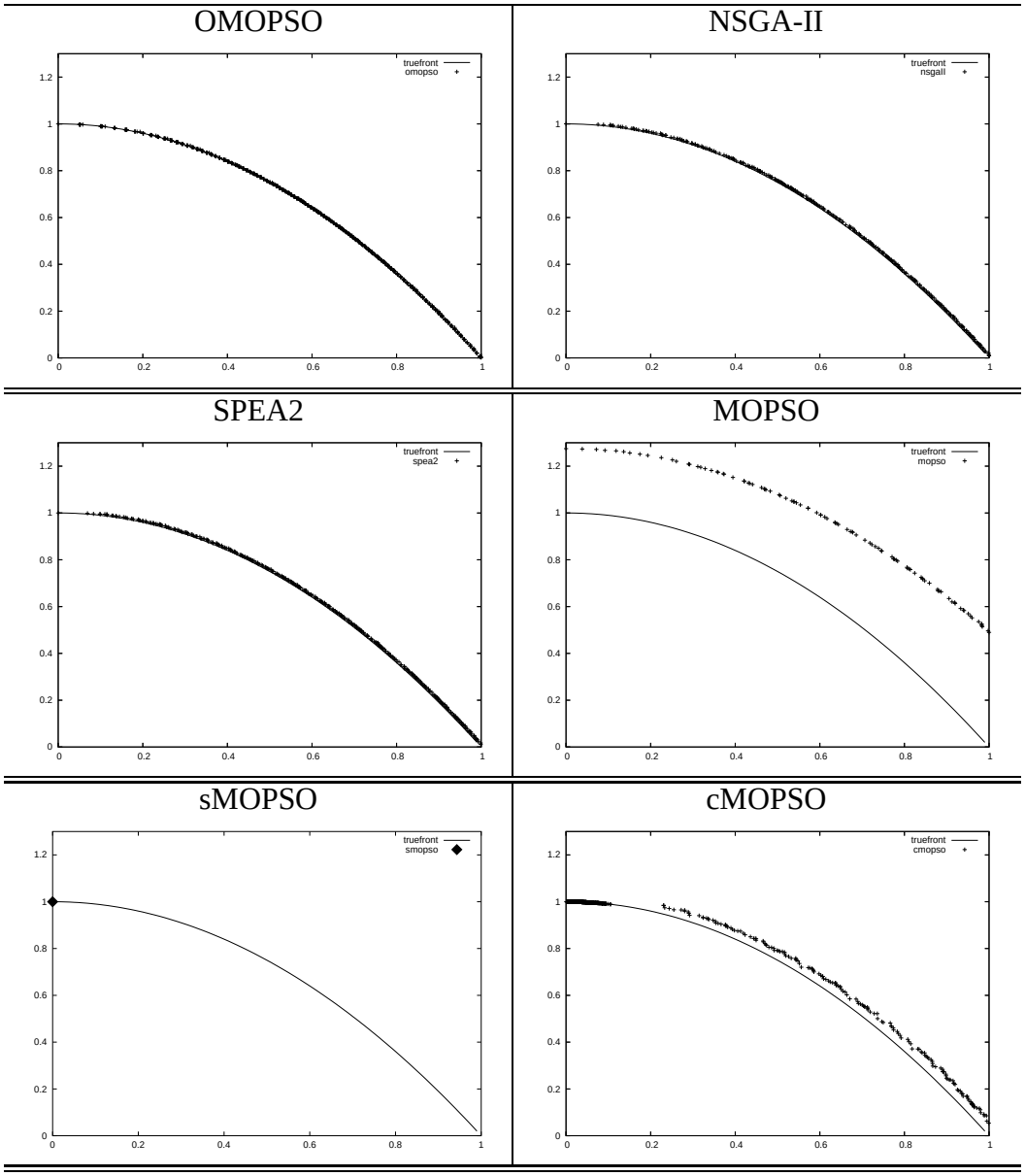


Figure A.3: Pareto fronts obtained by all the approaches for test function ZDT2. Our algorithm is denoted by OMOPSO and, in this case, it used ε=0.0075.

Test Function ZDT3 - Two Set Coverage Measure SC						
SC(X,	OMOPSO)	NSGA-II)	SPEA2)	MOPSO)	sMOPSO)	cMOPSO)
OMOPSO	0.00	**0.55**	**0.63**	**0.99**	**0.23**	**0.92**
NSGA-II	0.06	0.00	0.66	1.00	0.14	1.00
SPEA2	0.03	0.08	0.00	1.00	0.13	1.00
MOPSO	0.00	0.00	0.00	0.00	0.00	0.00
sMOPSO	0.13	0.42	0.41	0.99	0.00	0.75
cMOPSO	0.00	0.00	0.00	1.00	0.01	0.00
Test Function ZDT3 - Two Set Hypervolume Measure HV						
HV(X,	OMOPSO)	NSGA-II)	SPEA2)	MOPSO)	sMOPSO)	cMOPSO)
OMOPSO	0.000000	-0.004777	-0.008855	-0.509439	0.002317	-0.040068
NSGA-II	0.002496	0.000000	-0.002404	-0.503446	0.006138	-0.032923
SPEA2	0.000894	0.000072	0.000000	-0.500970	0.007052	-0.030447
MOPSO	0.001280	0.000000	0.000000	0.000000	0.000000	0.000000
sMOPSO	0.000967	-0.002485	-0.004047	-0.512069	0.000000	-0.028319
cMOPSO	0.000128	0.000000	0.000000	-0.470523	0.013227	0.000000

Table A.6: Comparison of results using the binary measures for test function ZDT3. Our algorithm is denoted by OMOPSO.

Test Function ZDT4							
		OMOPSO	NSGA-II	SPEA2	MOPSO	sMOPSO	cMOPSO
SCC	best	96	0	0	0	0	0
	median	88	0	0	0	0	0
	worst	35	0	0	0	0	0
	average	**80**	0	0	0	0	0
	std. dev.	16.3	0	0	0	0	0
IGD	best	0.0009	0.0126	0.0256	4.6415	0.1541	0.4203
	median	0.0010	0.1317	0.0811	12.407	0.7393	1.6404
	worst	0.0010	0.3219	0.3464	15.250	1.2865	4.1864
	average	**0.0010**	0.1508	0.1224	9.9195	0.7591	1.8621
	std. dev.	0.00003	0.0973	0.0943	4.0106	0.3147	0.9357

Table A.7: Comparison of results between our approach (denoted by OMOPSO), NSGA-II [27], SPEA2 [118], MOPSO [17], sMOPSO [68] and cMOPSO [103], for test function ZDT4 with respect to the unary measures.

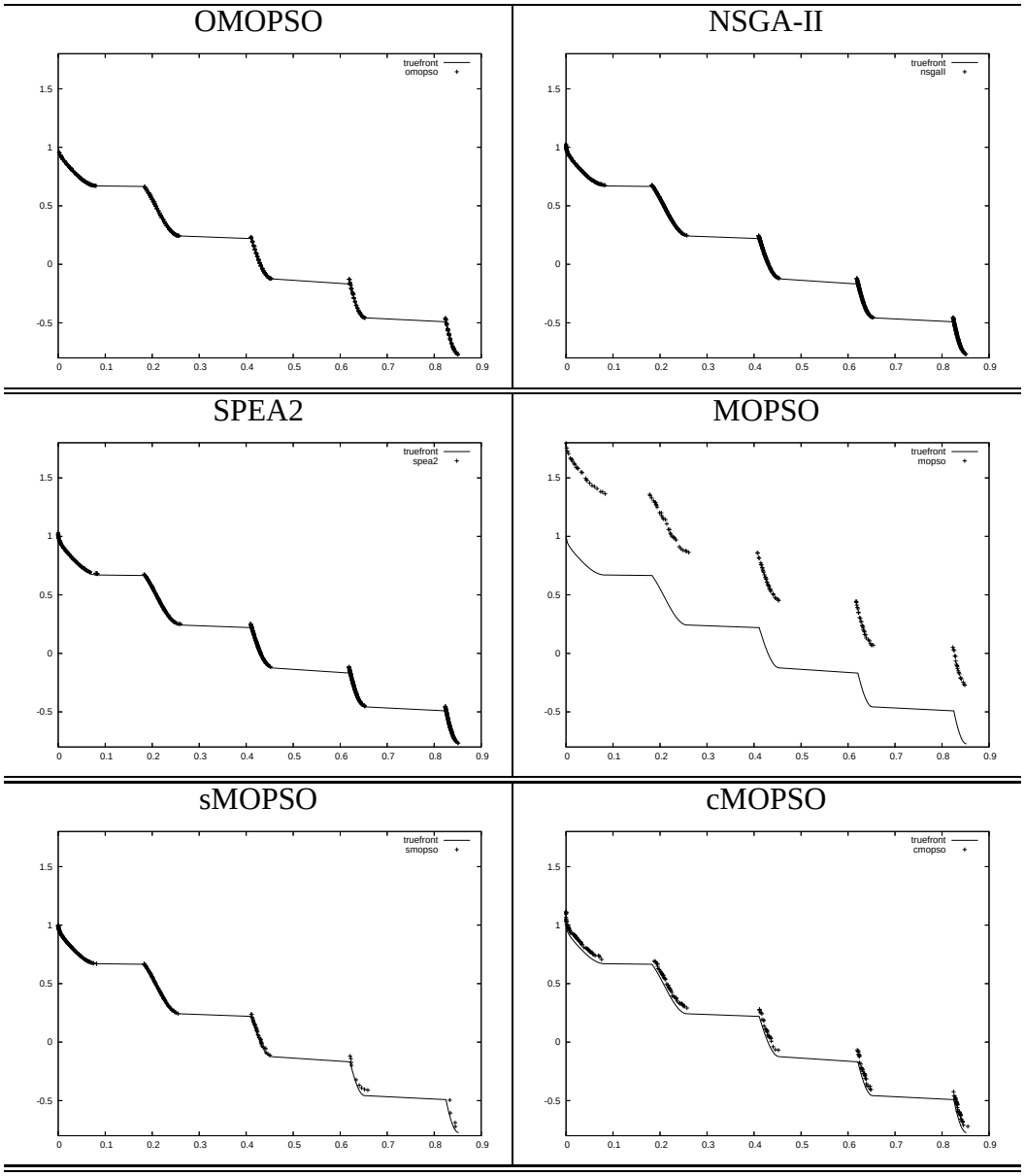


Figure A.4: Pareto fronts obtained by all the approaches for test function ZDT3. Our algorithm is denoted by OMOPSO and, in this case, it used ε=0.0026.

Test Function ZDT4 - Two Set Coverage Measure SC						
SC(X,	OMOPSO)	NSGA-II)	SPEA2)	MOPSO)	sMOPSO)	cMOPSO)
OMOPSO	0.00	**0.92**	**0.93**	0.00	0.00	0.00
NSGA-II	0.00	0.00	1.00	1.00	1.00	1.00
SPEA2	0.00	0.00	0.00	1.00	1.00	1.00
MOPSO	0.00	0.00	0.00	0.00	0.00	0.00
sMOPSO	0.00	0.00	0.00	1.00	0.00	1.00
cMOPSO	0.00	0.00	0.00	1.00	0.00	0.00
Test Function ZDT4 - Two Set Hypervolume Measure HV						
HV(X,	OMOPSO)	NSGA-II)	SPEA2)	MOPSO)	sMOPSO)	cMOPSO)
OMOPSO	0.000000	-0.163966	-0.342159	**-0.333024***	**-0.333024***	**-0.333024***
NSGA-II	0.000935	0.000000	-0.179995	-0.497925*	-0.497925*	-0.497925*
SPEA2	0.002737	0.000000	0.000000	-0.677920*	-0.677920*	-0.677920*
MOPSO	**0.000000**	0.000000	0.000000	0.000000	0.000000	0.000000
sMOPSO	**0.000000**	0.000000	0.000000	0.000000	0.000000	0.000000
cMOPSO	**0.000000**	0.000000	0.000000	0.000000	0.000000	0.000000

Table A.8: Comparison of results using the binary measures for test function ZDT4. Our algorithm is denoted by OMOPSO.

Test Function DTLZ2							
		OMOPSO	NSGA-II	SPEA2	MOPSO	sMOPSO	cMOPSO
SCC	best	31	24	26	98	32	31
	median	18	17	20	92	24	15
	worst	9	11	12	50	16	7
	average	18	16	20	**89**	25	16
	std. dev.	6.9	3.2	4.3	10.2	4.2	6.7
IGD	best	0.0014	0.0018	0.0013	0.0101	0.0013	0.0013
	median	0.0014	0.0020	0.0013	0.0118	0.0014	0.0021
	worst	0.0015	0.0025	0.0014	0.0213	0.0015	0.0027
	average	0.0014	0.0020	**0.0013**	0.0129	0.0014	0.0021
	std. dev.	0.00004	0.0002	0.00004	0.0030	0.00005	0.0004

Table A.9: Comparison of results between our approach (denoted by OMOPSO), NSGA-II [27], SPEA2 [118], MOPSO [17], sMOPSO [68] and cMOPSO [103], for test function DTLZ2 with respect to the unary measures.

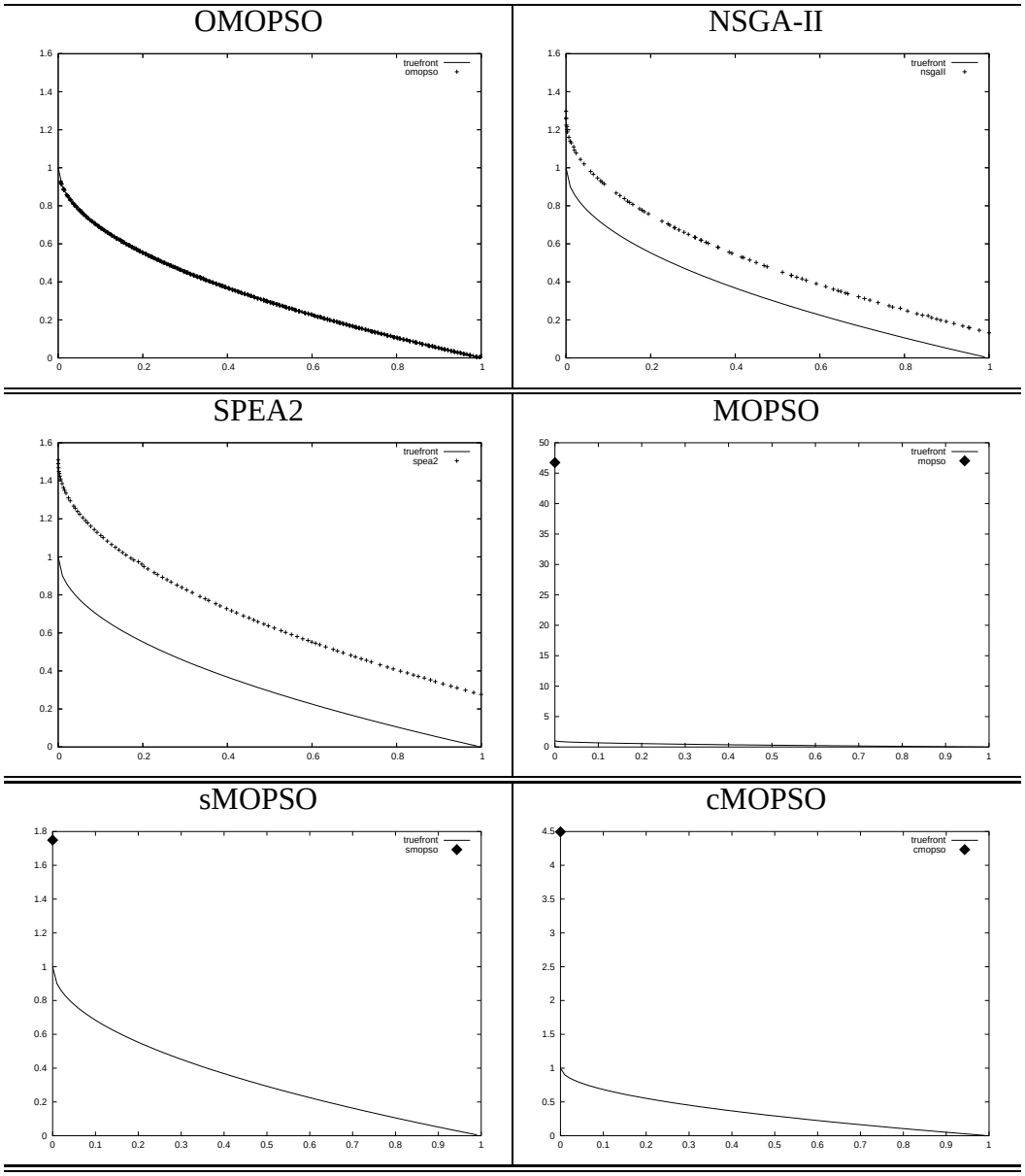


Figure A.5: Pareto fronts obtained by all the approaches for test function ZDT4. Our algorithm is denoted by OMOPSO and, in this case, it used ε=0.0075.

Test Function DTLZ2 - Two Set Coverage Measure SC						
SC(X,	OMOPSO)	NSGA-II)	SPEA2)	MOPSO)	sMOPSO)	cMOPSO)
OMOPSO	0.00	**0.22**	0.10	0.00	0.05	**0.55**
NSGA-II	0.04	0.00	0.06	0.00	0.03	0.51
SPEA2	**0.17**	0.39	0.00	0.00	0.07	0.64
MOPSO	**0.18**	0.24	0.21	0.00	0.21	0.40
sMOPSO	**0.28**	0.53	0.35	0.00	0.00	0.97
cMOPSO	0.03	0.08	0.04	0.00	0.00	0.00
Test Function DTLZ2 - Two Set Hypervolume Measure HV						
HV(X,	OMOPSO)	NSGA-II)	SPEA2)	MOPSO)	sMOPSO)	cMOPSO)
OMOPSO	0.000000	-0.003302	0.001478	-0.222739*	0.001736	-0.043070
NSGA-II	0.023198	0.000000	0.004201	-0.242413*	0.003977	-0.034526
SPEA2	0.017948	-0.005829	0.000000	-0.235493*	0.003266	-0.047018
MOPSO	0.000975	0.005851*	0.002741*	0.000000	0.002248*	0.013356*
sMOPSO	0.014884	-0.009375	-0.000056	-0.232664*	0.000000	-0.050838
cMOPSO	0.022044	0.004088	0.001626	-0.273522*	0.001128	0.000000

Table A.10: Comparison of results using the binary measures for test function DTLZ2. Our algorithm is denoted by OMOPSO.

Test Function DTLZ4							
		OMOPSO	NSGA-II	SPEA2	MOPSO	sMOPSO	cMOPSO
SCC	best	23	88	91	92	68	10
	median	11	20	18	34	42	0
	worst	2	13	11	6	8	0
	average	11	34	35	37	**39**	1.4
	std. dev.	4.4	27.1	32.8	22.9	14.8	3.3
IGD	best	0.0044	0.0016	0.0013	0.0026	0.0051	0.0074
	median	0.0106	0.0018	0.0014	0.0047	0.0068	0.0234
	worst	0.0168	0.0168	0.0168	0.0117	0.0073	0.0342
	average	0.0106	0.0047	**0.0045**	0.0060	0.0064	0.0223
	std. dev.	0.0034	0.0054	0.0057	0.0031	0.0007	0.0074

Table A.11: Comparison of results between our approach (denoted by OMOPSO), NSGA-II [27], SPEA2 [118], MOPSO [17], sMOPSO [68] and cMOPSO [103], for test function DTLZ4 with respect to the unary measures.

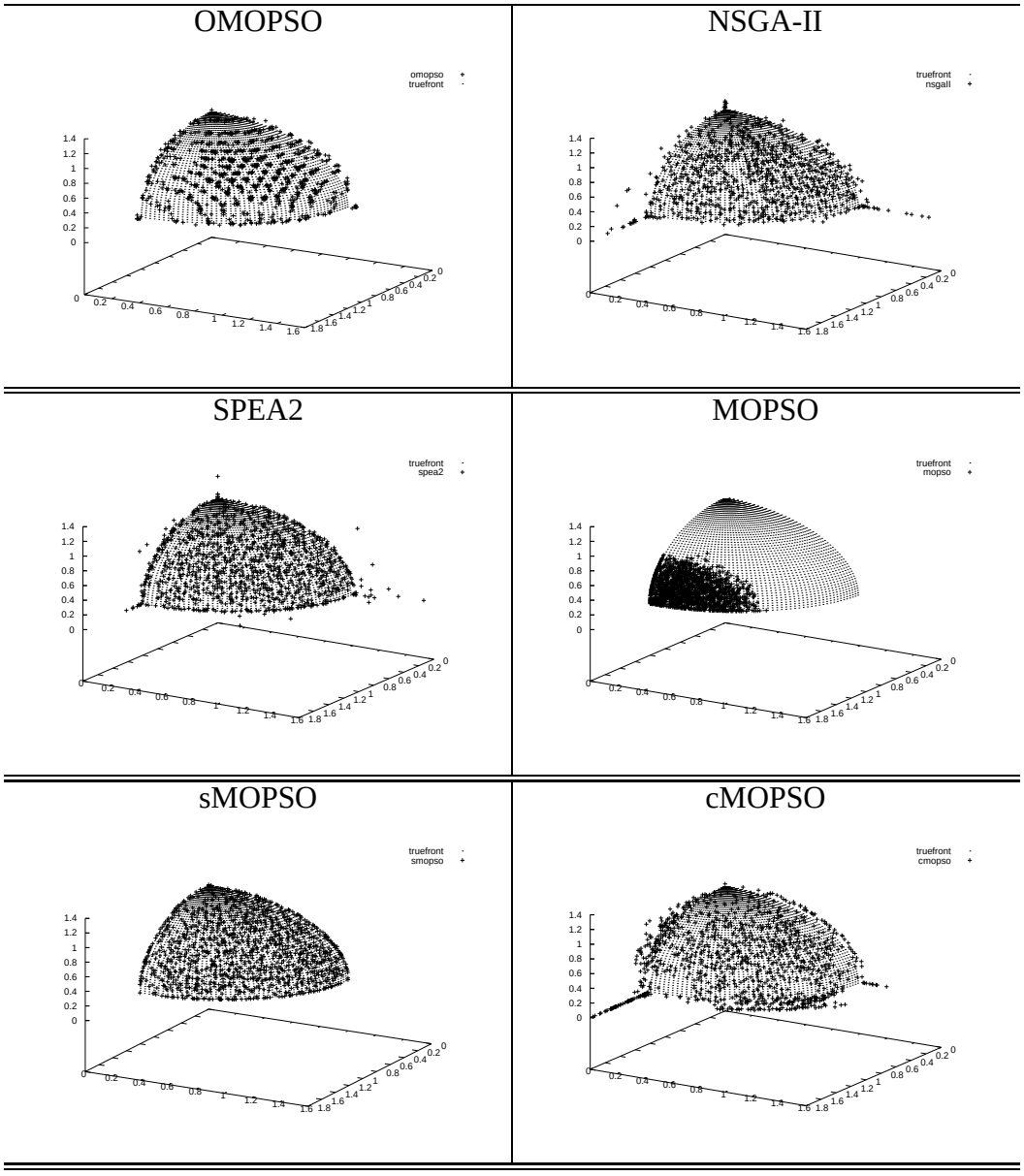


Figure A.6: Pareto fronts obtained by all the approaches for test function DTLZ2. Our algorithm is denoted by OMOPSO and, in this case, it used ε=0.066.

Test Function DTLZ4 - Two Set Coverage Measure SC						
SC(X,	OMOPSO)	NSGA-II)	SPEA2)	MOPSO)	sMOPSO)	cMOPSO)
OMOPSO	0.00	**0.08**	0.06	**0.14**	**0.17**	**0.98**
NSGA-II	0.04	0.00	0.10	0.24	0.25	1.00
SPEA2	**0.07**	0.19	0.00	0.28	0.25	1.00
MOPSO	0.06	0.14	0.11	0.00	0.27	0.94
sMOPSO	0.06	0.09	0.08	0.10	0.00	1.00
cMOPSO	0.00	0.00	0.00	0.00	0.00	0.00
Test Function DTLZ4 - Two Set Hypervolume Measure HV						
HV(X,	OMOPSO)	NSGA-II)	SPEA2)	MOPSO)	sMOPSO)	cMOPSO)
OMOPSO	0.000000	0.000893*	0.000467*	0.000125	0.166099	-0.221553
NSGA-II	- 0.304934*	0.000000	0.005186	-0.046700*	-0.430067*	-0.527380*
SPEA2	- 0.301975*	0.001801	0.000000	-0.043671*	-0.426669*	-0.523995*
MOPSO	- 0.259236*	0.000016*	0.000341	0.000000	-0.383423*	-0.480664*
sMOPSO	0.041476	0.000383*	0.000396*	0.000311*	0.000000	0.000000
cMOPSO	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

Table A.12: Comparison of results using the binary measures for test function DTLZ4. Our algorithm is denoted by OMOPSO.

Test Function DTLZ6							
		OMOPSO	NSGA-II	SPEA2	MOPSO	sMOPSO	cMOPSO
SCC	best	84	1	2	0	1	0
	median	61	0	0	0	1	0
	worst	33	0	0	0	1	0
	average	**61**	0.1	0.6	0	1	0
	std. dev.	12.6	0.3	0.9	0	0	0
IGD	best	0.0024	0.0064	0.0037	0.0375	0.0673	0.0110
	median	0.0138	0.0088	0.0045	0.0583	0.0673	0.0345
	worst	0.0151	0.0314	0.0214	0.1185	0.0673	0.0742
	average	0.0096	0.0132	**0.0067**	0.0658	0.0673	0.0373
	std. dev.	0.0058	0.0083	0.0051	0.0205	0.0000	0.0172

Table A.13: Comparison of results between our approach (denoted by OMOPSO), NSGA-II [27], SPEA2 [118], MOPSO [17], sMOPSO [68] and cMOPSO [103], for test function DTLZ6 with respect to the unary measures.

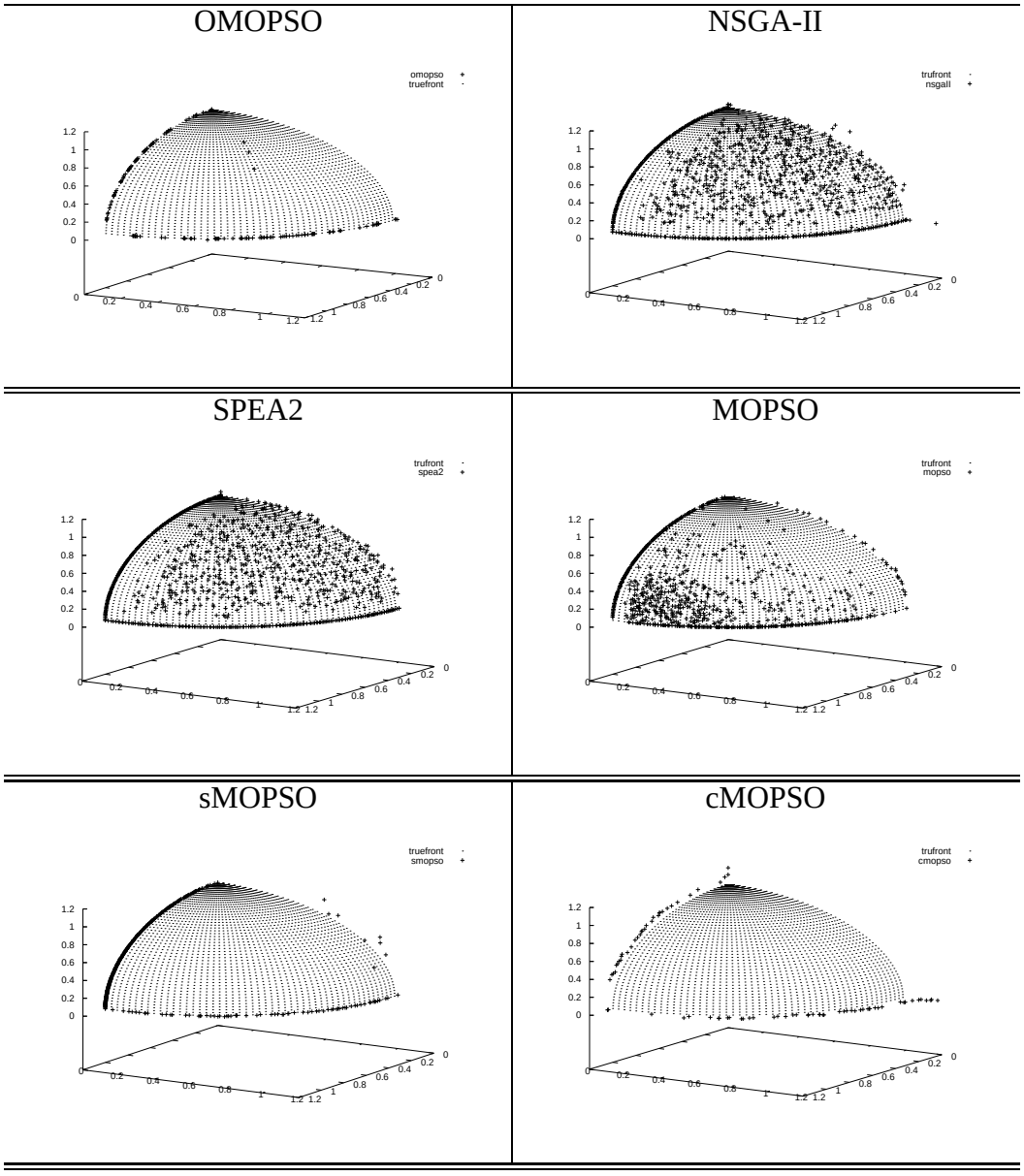


Figure A.7: Pareto fronts obtained by all the approaches for test function DTLZ4. Our algorithm is denoted by OMOPSO and, in this case, it used ε=0.059.

Test Function DTLZ6 - Two Set Coverage Measure SC						
SC(X,	OMOPSO)	NSGA-II)	SPEA2)	MOPSO)	sMOPSO)	cMOPSO)
OMOPSO	0.00	**0.72**	**0.73**	**0.87**	0.00	**0.57**
NSGA-II	0.00	0.00	0.31	1.00	0.00	0.80
SPEA2	0.00	0.30	0.00	0.98	0.00	0.57
MOPSO	0.00	0.00	0.00	0.00	0.00	0.00
sMOPSO	0.00	0.07	0.12	0.45	0.00	0.24
cMOPSO	0.00	0.04	0.12	1.00	0.00	0.00
Test Function DTLZ6 - Two Set Hypervolume Measure HV						
HV(X,	OMOPSO)	NSGA-II)	SPEA2)	MOPSO)	sMOPSO)	cMOPSO)
OMOPSO	0.000000	-0.171358	-0.174556	-0.340801	**-3.597268***	-0.638663
NSGA-II	0.057354	0.000000	0.001996	-0.113803	-3.825408*	-0.410346
SPEA2	0.071737	0.019577	0.000000	-0.096222	-3.843163*	-0.392616
MOPSO	0.001714	0.000000	0.000000	0.000000	-3.880365*	0.000000
sMOPSO	**0.000000**	0.000572*	0.000398*	0.059418*	0.000000	0.001163*
cMOPSO	0.000286	-0.000109	0.000040	0.296434	-4.235054*	0.000000

Table A.14: Comparison of results using the binary measures for test function DTLZ6. Our algorithm is denoted by OMOPSO.

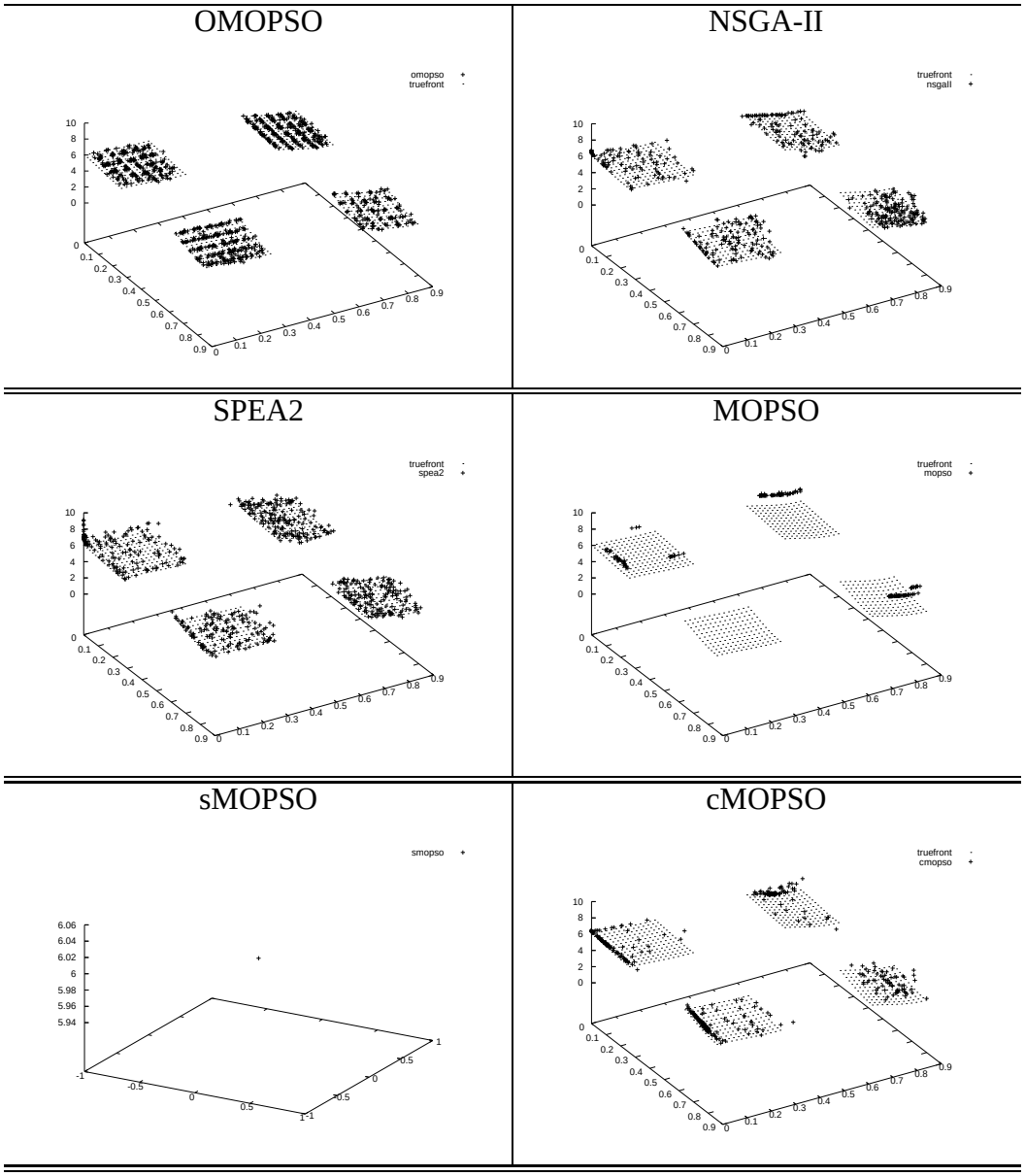


Figure A.8: Pareto fronts obtained by all the approaches for test function DTLZ6. Our algorithm is denoted by OMOPSO and, in this case, it used ε=0.05.

Appendix B

Results provided by the ANOVA performed and the adaptation mechanisms proposed

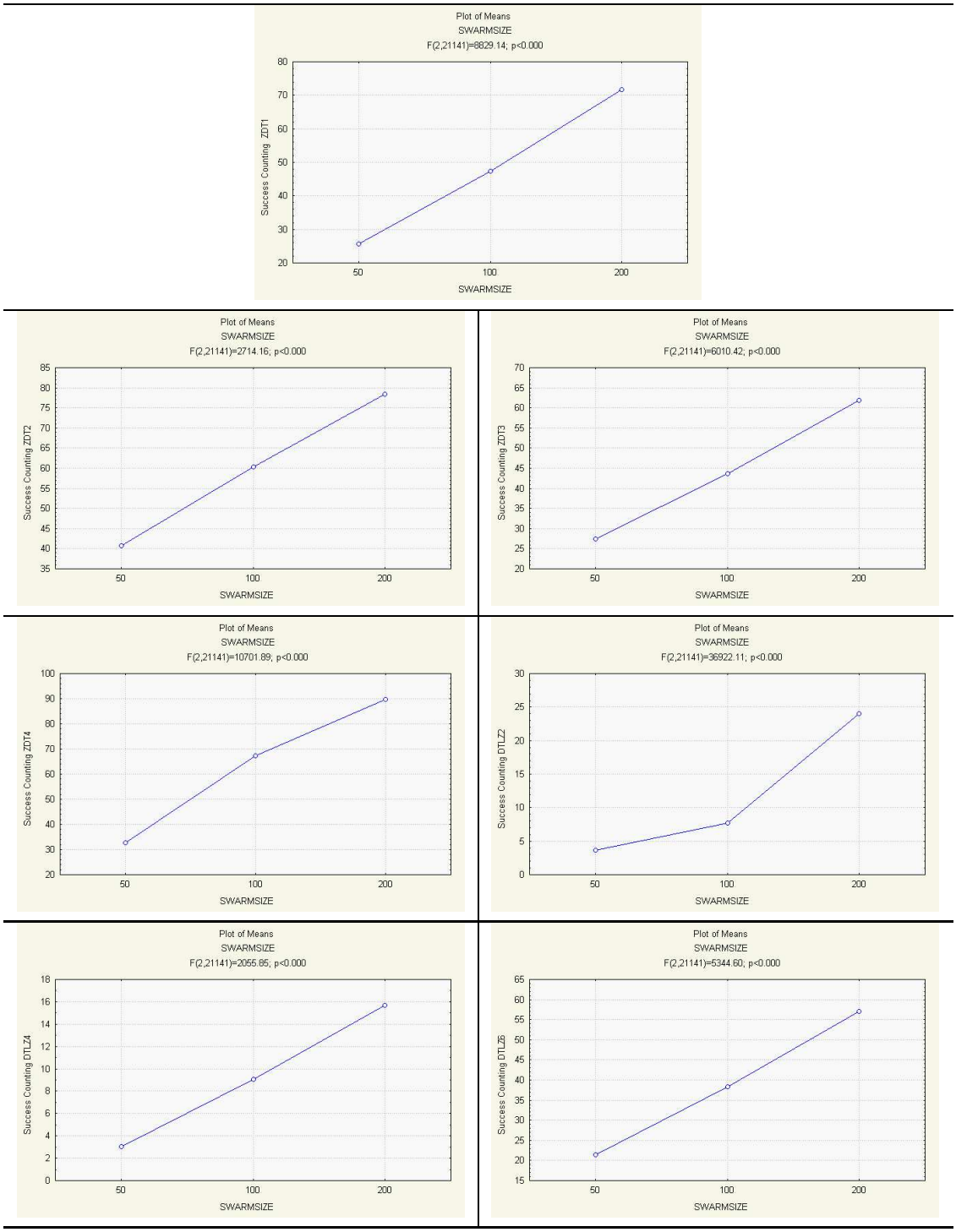


Figure B.1: Results obtained from the ANOVA, for the *swarmsize* parameter.

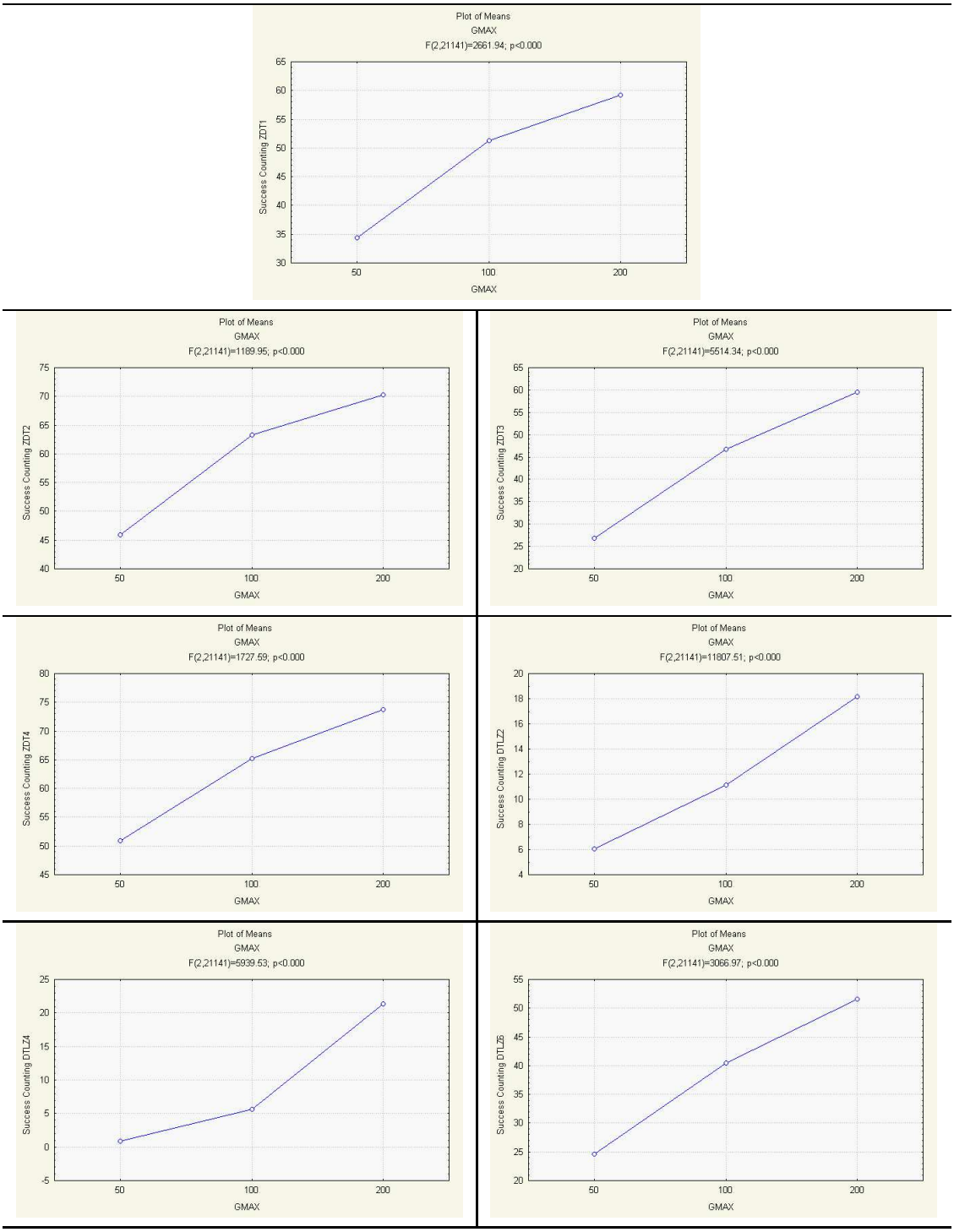


Figure B.2: Results obtained from the ANOVA, for the *gmax* parameter.

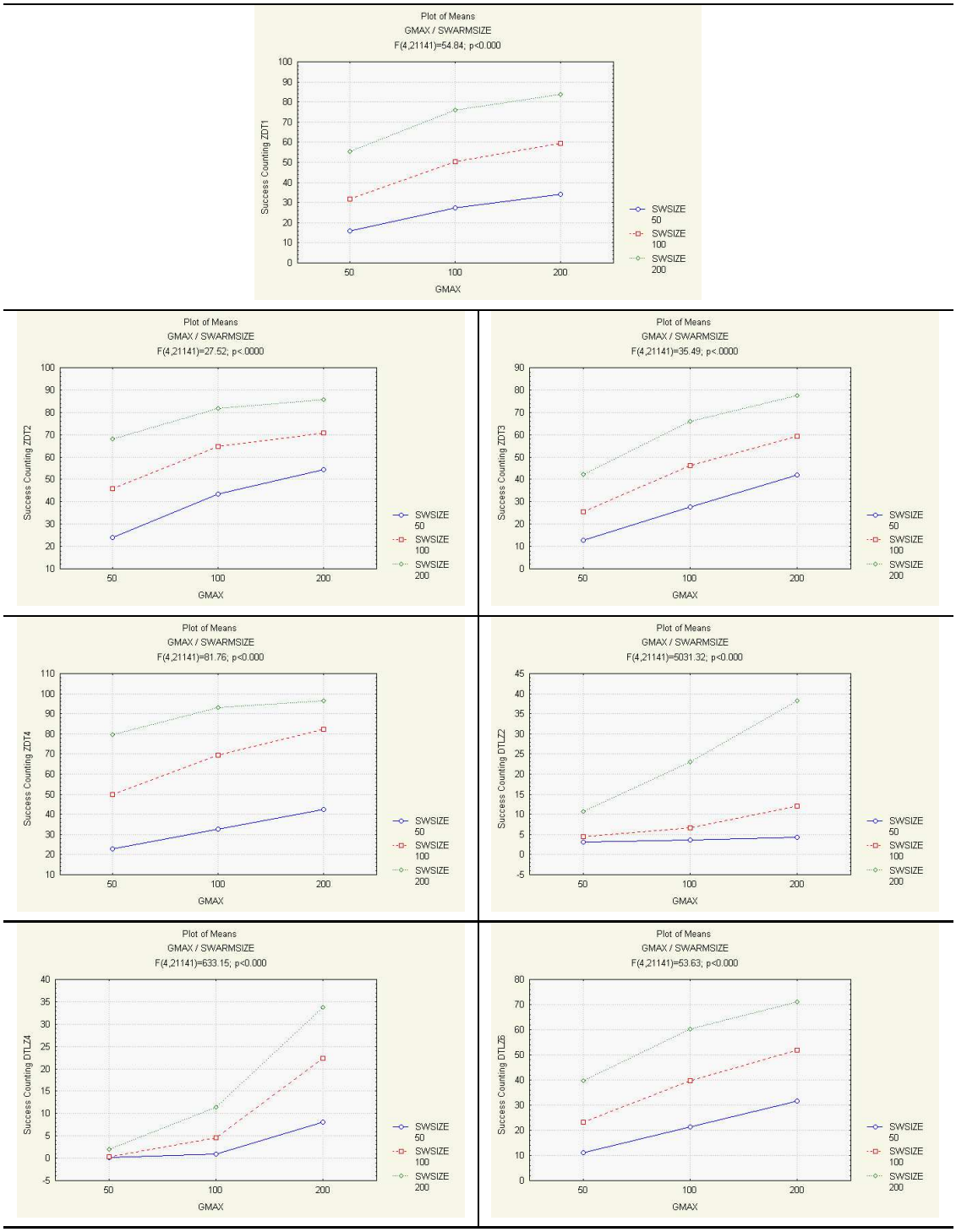


Figure B.3: Correlation observed between parameters *swarmsize* and *gmax*.

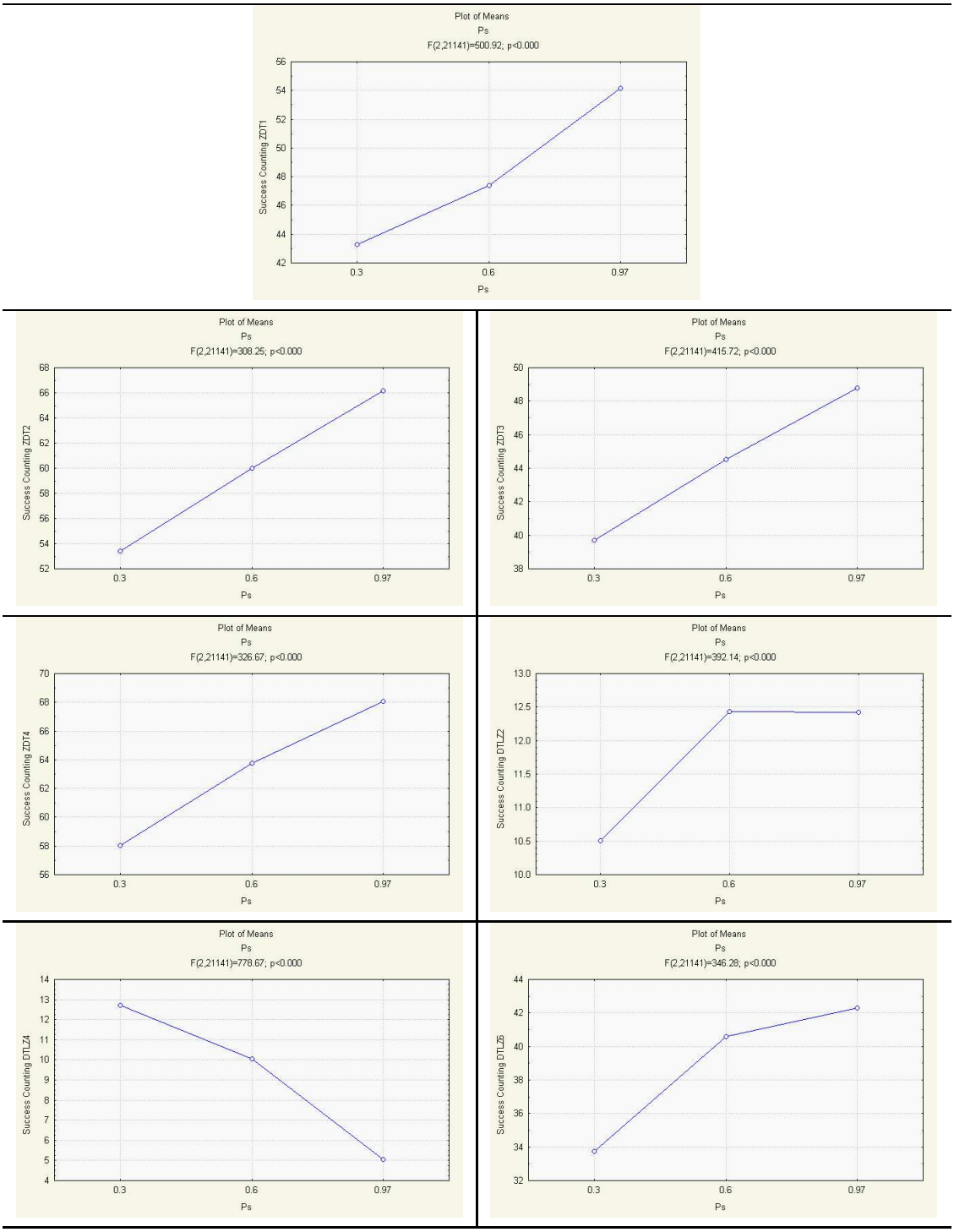


Figure B.4: Results obtained from the ANOVA, for the *Ps* parameter.

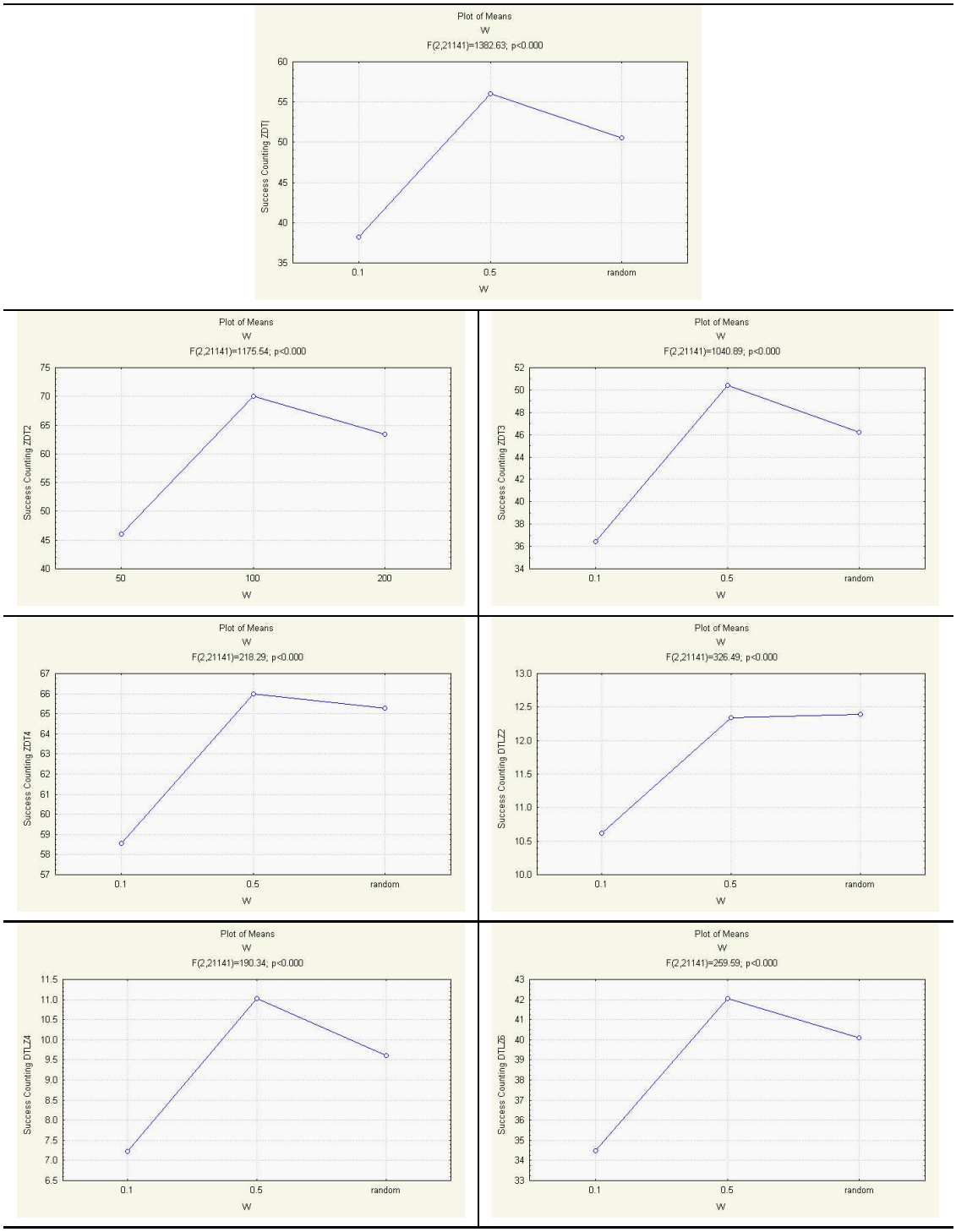


Figure B.5: Results obtained from the ANOVA, for the *W* parameter.

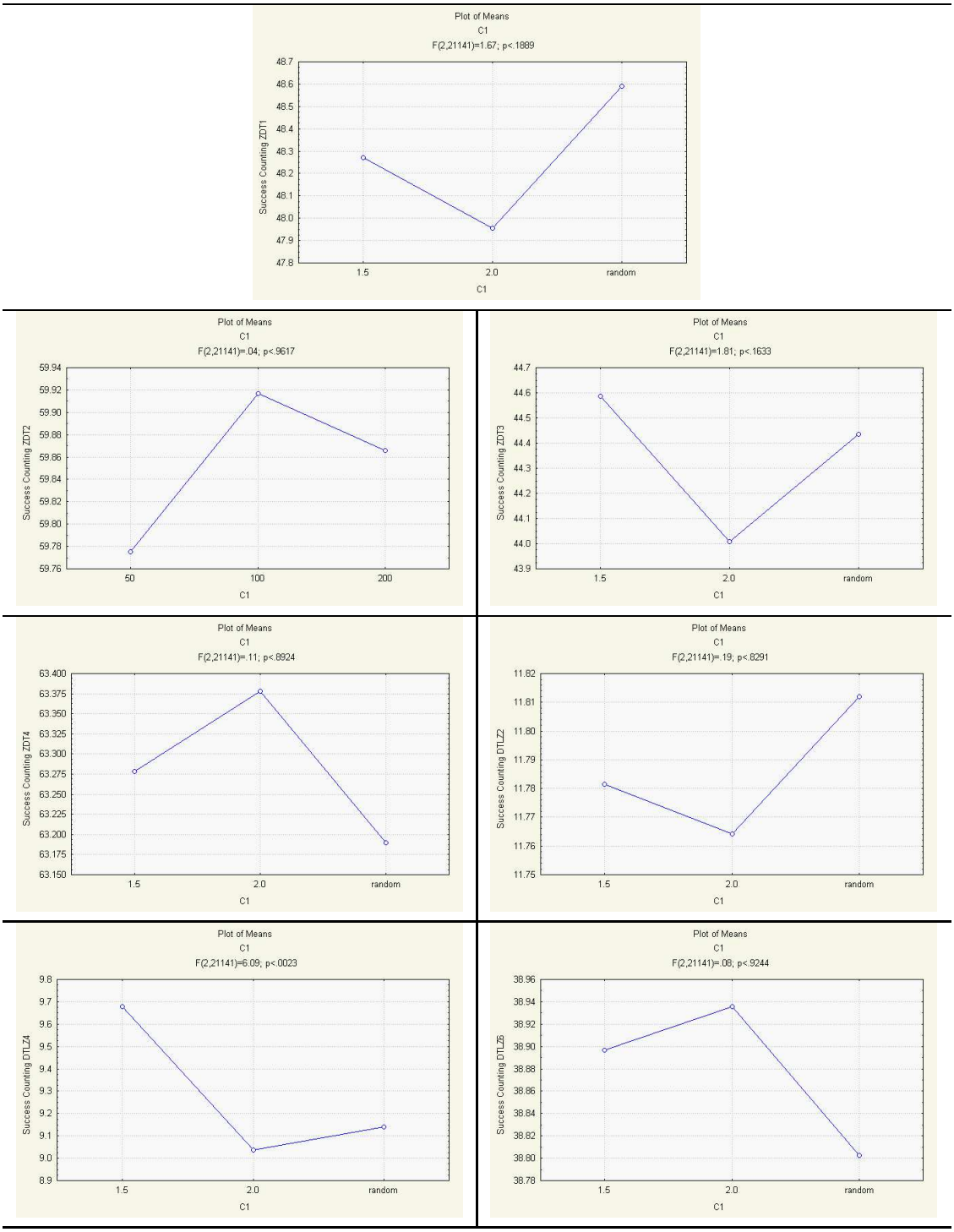


Figure B.6: Results obtained from the ANOVA, for the C_1 parameter.

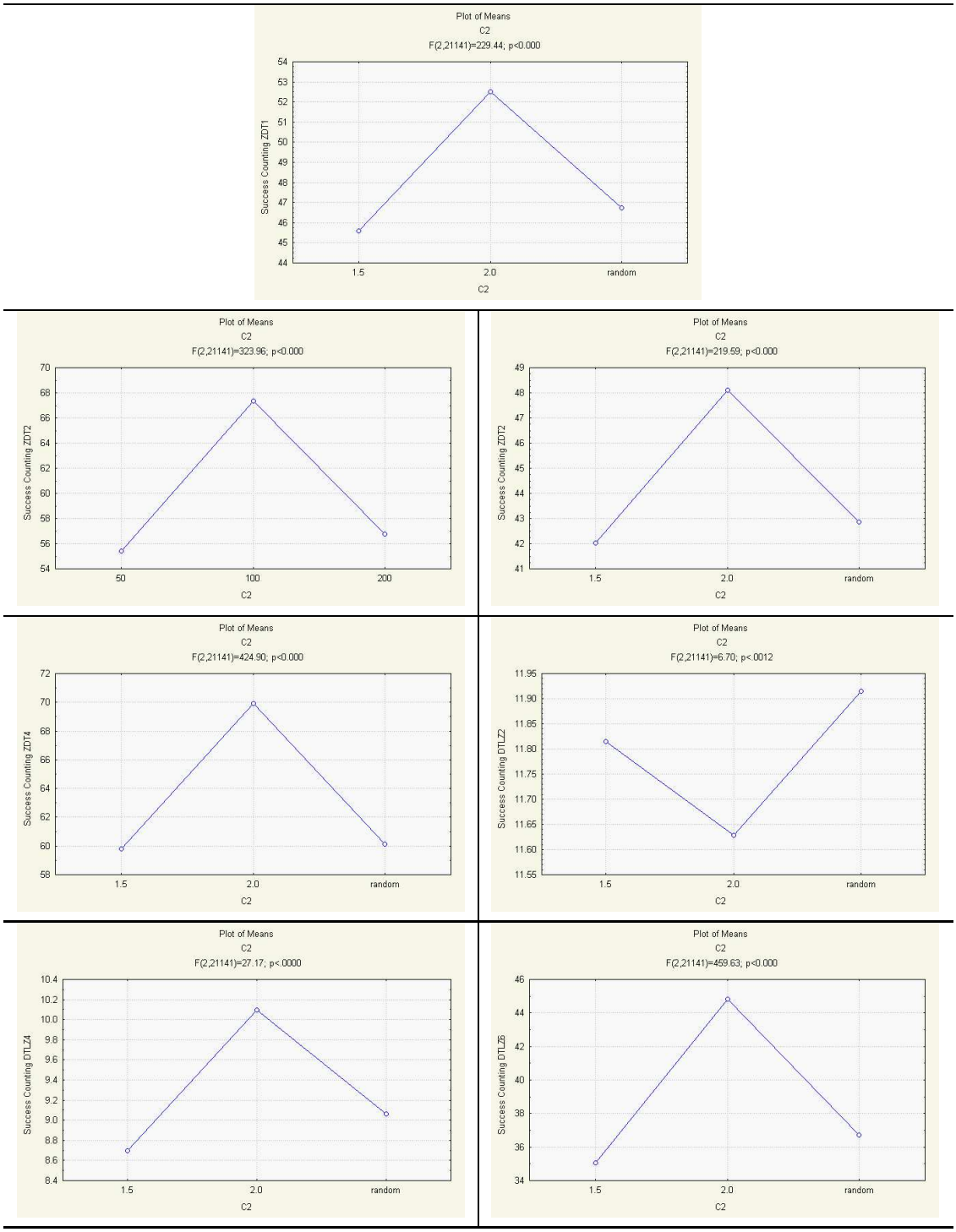


Figure B.7: Results obtained from the ANOVA, for the C_2 parameter.

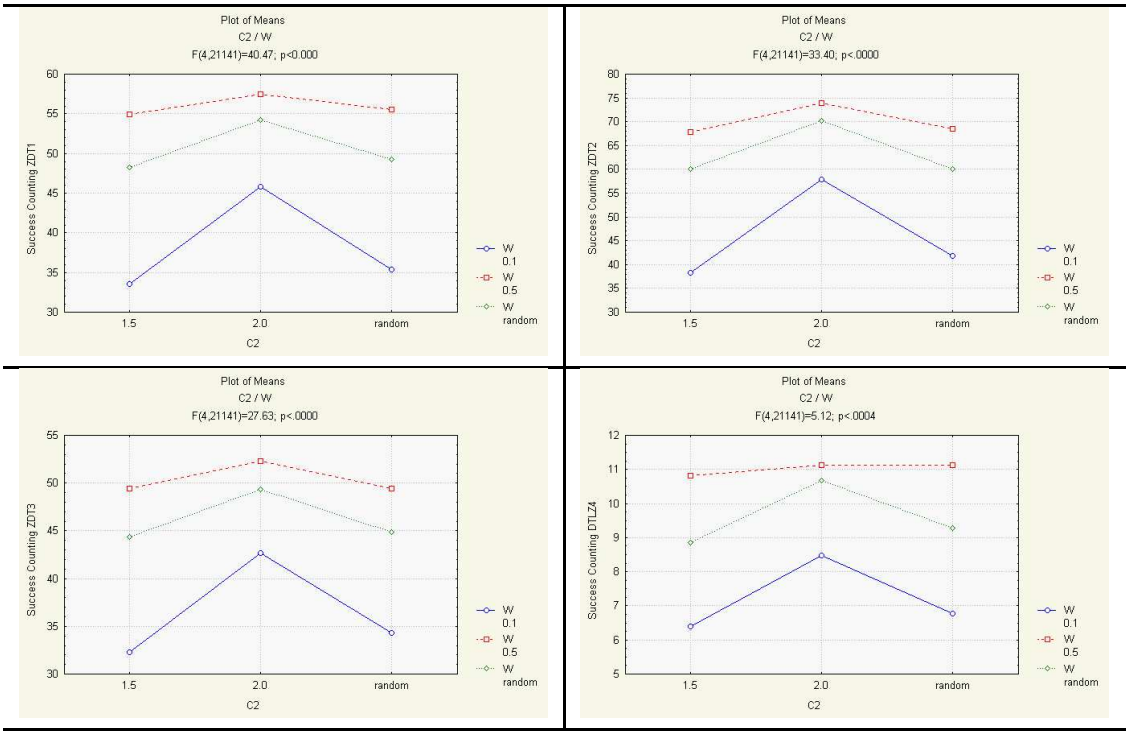


Figure B.8: Correlation observed between parameters W and C_2.

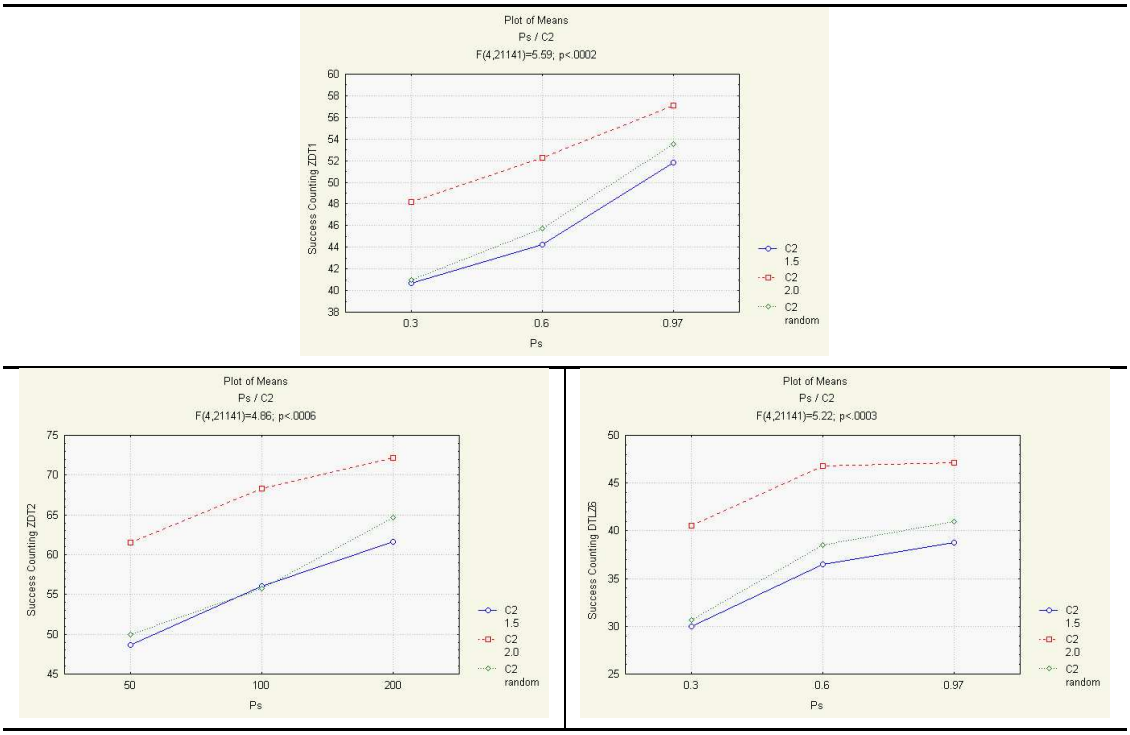


Figure B.9: Correlation observed between parameters C_2 and Ps.

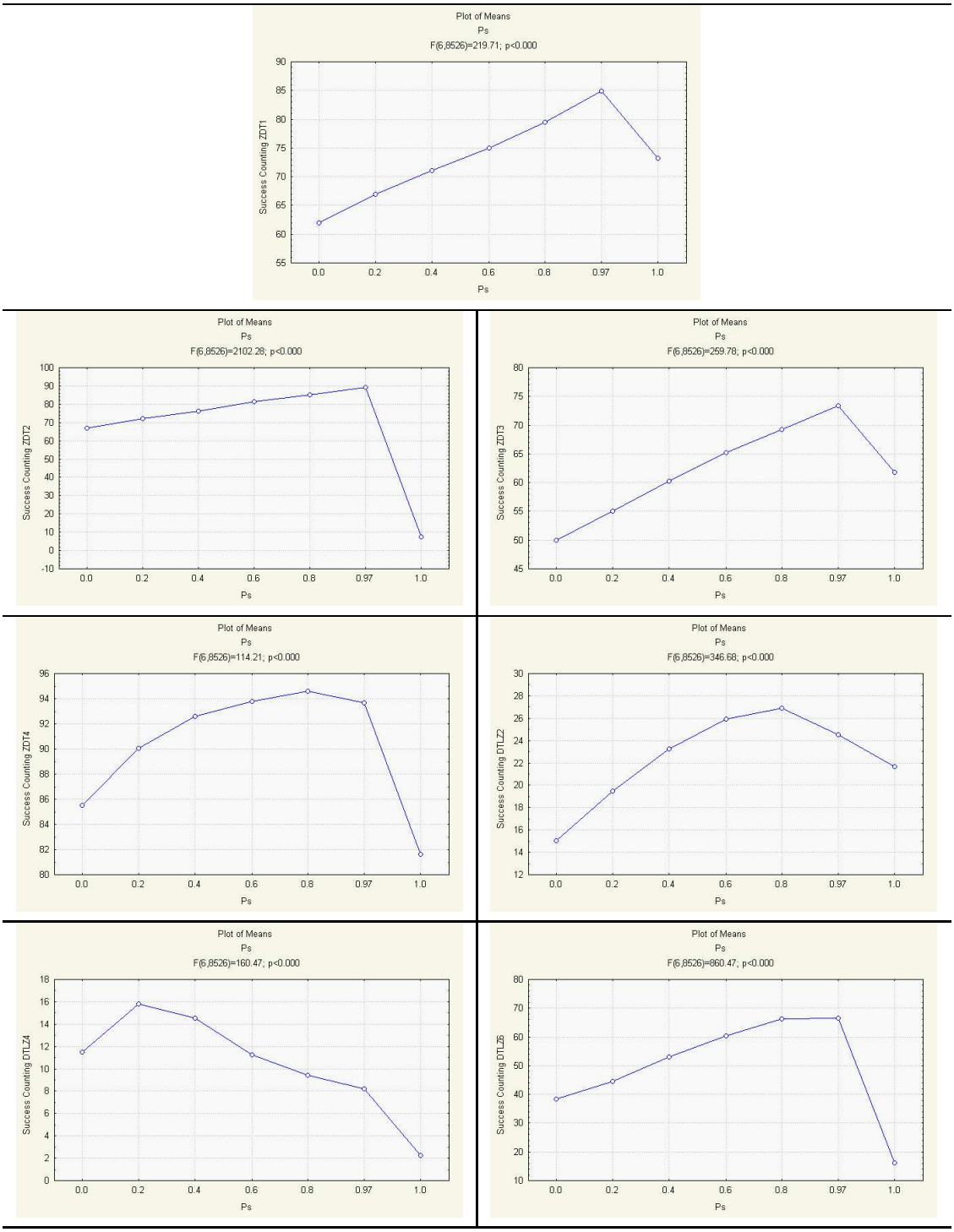


Figure B.10: Results obtained from the second ANOVA, for the *Ps* parameter.

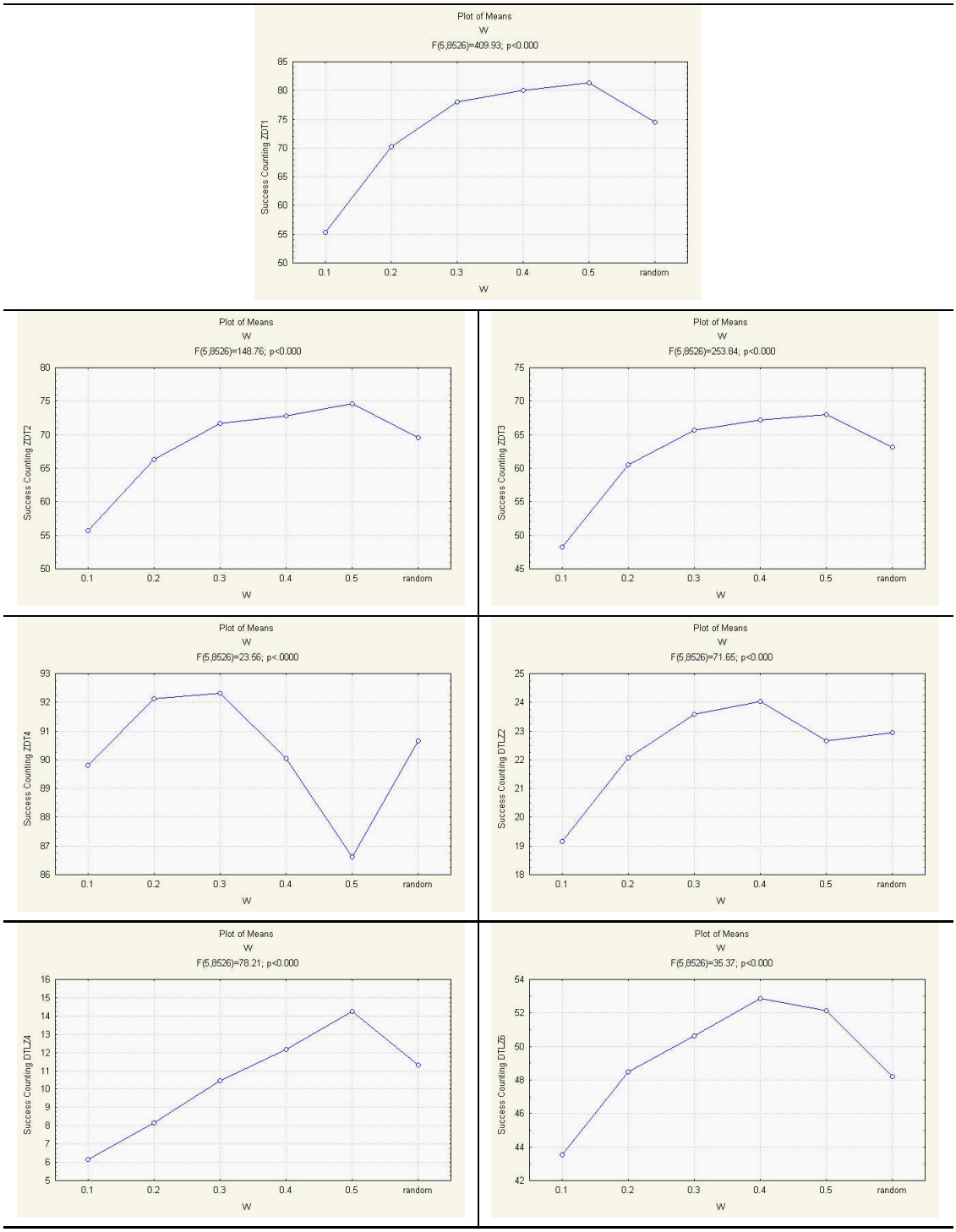


Figure B.11: Results obtained from the second ANOVA, for the *W* parameter.

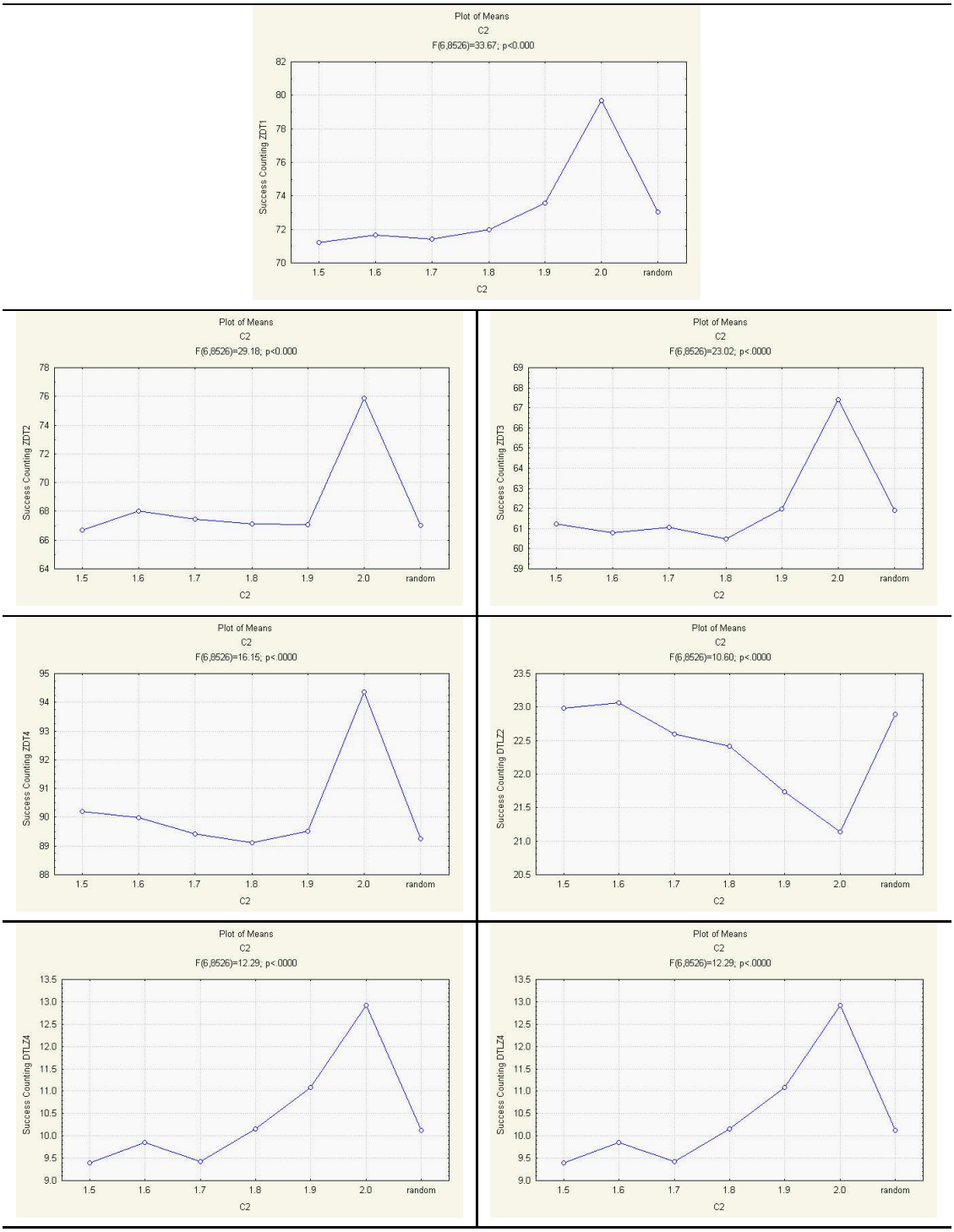


Figure B.12: Results obtained from the second ANOVA, for the C_2 parameter.

Function ZDT1								
SCC	without	with adaptation and reward 1						
measure	adaptation	Prop.	$\varepsilon(0.05)$	$\varepsilon(0.10)$	$\varepsilon(0.15)$	$\tau(0.05)$	$\tau(0.10)$	$\tau(0.15)$
best	100	99	99	98	99	99	100	97
median	86	82	82	81	85	84	83	85
worst	15	51	30	18	20	48	58	62
mean	84	81	78	80	82	82	82	84
st.dev.	17.5	9.1	17.7	15.0	15.3	11.9	9.9	9.5
SCC	without	with adaptation and reward 2						
measure	adaptation	Prop.	$\varepsilon(0.05)$	$\varepsilon(0.10)$	$\varepsilon(0.15)$	$\tau(0.05)$	$\tau(0.10)$	$\tau(0.15)$
best	100	100	98	93	93	97	98	96
median	86	95	76	72	77	82	81	83
worst	15	54	19	27	29	49	15	60
mean	84	92	70	68	72	82	80	82
st.dev.	17.5	9.8	19.6	17.2	17.3	9.9	15.8	9.7

Table B.1: Results obtained for function ZDT1.

Function ZDT2								
SCC	without	with adaptation and reward 1						
measure	adaptation	Prop.	$\varepsilon(0.05)$	$\varepsilon(0.10)$	$\varepsilon(0.15)$	$\tau(0.05)$	$\tau(0.10)$	$\tau(0.15)$
best	100	100	100	100	100	100	100	100
median	96	93	93	94	95	90	93	92
worst	60	16	23	58	45	34	30	0
mean	94	89	84	89	89	83	86	87
st.dev.	7.8	15.9	18.4	10.6	13.4	15.4	16.9	18.1
SCC	without	with adaptation and reward 2						
measure	adaptation	Prop.	$\varepsilon(0.05)$	$\varepsilon(0.10)$	$\varepsilon(0.15)$	$\tau(0.05)$	$\tau(0.10)$	$\tau(0.15)$
best	100	100	100	100	100	100	100	100
median	96	98	82	92	91	93	94	95
worst	60	54	12	0	22	24	44	1
mean	94	94	76	77	81	90	87	88
st.dev.	7.8	9.5	23.7	28.2	20.2	13.9	15.5	18.6

Table B.2: Results obtained for function ZDT2.

Function ZDT3								
SCC	without	with adaptation and reward 1						
measure	adaptation	Prop.	$\varepsilon(0.05)$	$\varepsilon(0.10)$	$\varepsilon(0.15)$	$\tau(0.05)$	$\tau(0.10)$	$\tau(0.15)$
best	95	92	90	88	92	93	91	91
median	83	74	74	78	76	74	78	74
worst	63	9	41	51	25	30	16	39
mean	81	69	74	75	72	74	74	72
st.dev.	9.8	19.5	10.8	9.7	14.8	16.5	15.7	10.6
SCC	without	with adaptation and reward 2						
measure	adaptation	Prop.	$\varepsilon(0.05)$	$\varepsilon(0.10)$	$\varepsilon(0.15)$	$\tau(0.05)$	$\tau(0.10)$	$\tau(0.15)$
best	95	97	83	81	83	92	91	93
median	83	86	63	60	60	75	76	72
worst	63	46	13	22	21	32	33	25
mean	81	83	59	56	55	73	73	68
st.dev.	9.8	12.5	15.7	14.5	17.5	11.5	14.8	18.9

Table B.3: Results obtained for function ZDT3.

Function ZDT4								
SCC	without	with adaptation and reward 1						
measure	adaptation	Prop.	$\varepsilon = 0.05$	$\varepsilon(0.10)$	$\varepsilon(0.15)$	$\tau(0.05)$	$\tau(0.10)$	$\tau(0.15)$
best	99	100	100	100	100	100	99	99
median	97	97	97	97	95	99	97	97
worst	73	94	81	88	85	86	89	89
mean	95	97	95	96	95	98	97	96
st.dev.	5.2	1.5	4.4	3.4	3.7	2.7	2.1	2.3
SCC	without	with adaptation and reward 2						
measure	adaptation	Prop.	$\varepsilon(0.05)$	$\varepsilon(0.10)$	$\varepsilon(0.15)$	$\tau(0.05)$	$\tau(0.10)$	$\tau(0.15)$
best	99	98	100	100	100	100	100	100
median	97	95	97	98	98	97	97	97
worst	73	84	0	89	91	84	88	88
mean	95	94	92	97	97	96	96	96
st.dev.	5.2	2.8	17.9	2.7	2.6	3.2	2.9	2.6

Table B.4: Results obtained for function ZDT4.

Function DTLZ2								
SCC	without	with adaptation and reward 1						
measure	adaptation	Prop.	ε(0.05)	ε(0.10)	ε(0.15)	τ(0.05)	τ(0.10)	τ(0.15)
best	50	41	38	48	38	51	40	47
median	28	29	28	26	27	32	27	30
worst	9	13	12	11	16	14	15	10
mean	30	28	27	26	26	32	27	28
st.dev.	9.6	7.4	6.9	8.1	5.9	9.5	6.9	8.4
SCC	without	with adaptation and reward 2						
measure	adaptation	Prop.	ε(0.05)	ε(0.10)	ε(0.15)	τ(0.05)	τ(0.10)	τ(0.15)
best	50	34	31	35	37	47	42	43
median	28	22	20	17	18	28	28	30
worst	9	8	8	5	5	7	13	8
mean	30	21	20	19	19	28	29	28
st.dev.	9.6	6.2	6.7	8.6	6.8	9.3	7.8	7.7

Table B.5: Results obtained for function DTLZ2.

Function DTLZ4								
SCC	without	with adaptation and reward 1						
measure	adaptation	Prop.	ε(0.05)	ε(0.10)	ε(0.15)	τ(0.05)	τ(0.10)	τ(0.15)
best	17	58	86	51	77	44	63	27
median	9	9	10	10	12	10	10	12
worst	1	1	1	1	2	1	1	2
mean	9	11	18	11	17	11	14	12
st.dev.	4.7	10.9	22	11.3	17.4	8.3	13.5	4.4
SCC	without	with adaptation and reward 2						
measure	adaptation	Prop.	ε(0.05)	ε(0.10)	ε(0.15)	τ(0.05)	τ(0.10)	τ(0.15)
best	17	19	49	75	83	26	61	58
median	9	9	8	7	9	8	9	11
worst	1	2	1	0	0	0	0	1
mean	9	8	11	9	11	9	11	13
st.dev.	4.7	3.9	12.3	13.9	15.8	6.1	13.7	10.2

Table B.6: Results obtained for function DTLZ4.

Function DTLZ6								
SCC	without	with adaptation and reward 1						
measure	adaptation	Prop.	ε(0.05)	ε(0.10)	ε(0.15)	τ(0.05)	τ(0.10)	τ(0.15)
best	97	96	100	92	100	98	100	100
median	64	62	66	56	59	68	71	75
worst	43	18	38	23	29	33	38	27
mean	67	62	68	57	60	69	73	71
st.dev.	14.9	14.9	16.6	18.5	16.1	21.1	17.1	21.5
SCC	without	with adaptation and reward 2						
measure	adaptation	Prop.	ε(0.05)	ε(0.10)	ε(0.15)	τ(0.05)	τ(0.10)	τ(0.15)
best	97	94	96	96	96	100	100	97
median	64	68	54	49	46	72	73	76
worst	43	27	9	2	6	18	21	14
mean	67	65	52	48	51	69	67	69
st.dev.	14.9	17.7	21.9	25.8	22.2	22.6	23.2	21.9

Table B.7: Results obtained for function DTLZ6.

Appendix C

Results provided by the fi tness inheritance techniques proposed

Test Function ZDT1							
		sMOPSO	oMOPSO	0.1	0.2	0.3	0.4
SCC	best	93	84	94	94	78	86
	median	58	74	79	72	69	65
	worst	23	22	44	27	25	36
	mean	59	71	**77**	64	62	61
	std. dev.	24.2	13.6	14.5	19.8	16.4	14.8
IGD	best	0.0031	0.0009	0.0009	0.0009	0.0009	0.0009
	median	0.0260	0.0010	0.0009	0.0010	0.0010	0.0010
	worst	0.0448	0.0011	0.0010	0.0011	0.0012	0.0087
	mean	0.0269	0.0010	**0.0009**	0.0010	0.0010	0.0014
	std. dev.	0.0095	0.00005	0.00003	0.00005	0.00008	0.0017

Test Function ZDT2							
		sMOPSO	oMOPSO	0.1	0.2	0.3	0.4
SCC	best	1	100	100	100	100	97
	median	1	91	93	92	88	84
	worst	1	60	6	54	23	33
	mean	1	**89**	83	86	79	77
	std. dev.	0	10	22.5	13.1	22.4	20.7
IGD	best	0.0723	0.0006	0.0006	0.0006	0.0006	0.0006
	median	0.0723	0.0007	0.0007	0.0007	0.0007	0.0007
	worst	0.0723	0.0008	0.0011	0.0009	0.0010	0.0009
	mean	0.0723	**0.0007**	0.0007	0.0007	0.0007	0.0007
	std. dev.	0.0000	0.00005	0.0001	0.00007	0.0001	0.00008

Table C.1: Results obtained for functions ZDT1 and ZDT2, for sMOPSO, oMOPSO, and oMOPSO with fitness inheritance (p_i=0.1,0.2,0.3,0.4).

Test Function ZDT3							
		sMOPSO	oMOPSO	0.1	0.2	0.3	0.4
SCC	best	89	90	88	82	85	92
	median	15	72	77	66	67	67
	worst	0	18	51	30	27	8
	mean	26	68	**73**	65	64	59
	std. dev.	25.4	18.2	11.2	13.9	14.5	21.2
IGD	best	0.0023	0.0008	0.0008	0.0008	0.0008	0.0008
	median	0.0249	0.0008	0.0008	0.0009	0.0008	0.0009
	worst	0.0374	0.0021	0.0106	0.0103	0.0014	0.0049
	mean	0.0245	0.0009	0.0015	0.0014	**0.0009**	0.0016
	std. dev.	0.0095	0.0003	0.0022	0.0021	0.0002	0.0012

Test Function ZDT4							
		sMOPSO	oMOPSO	0.1	0.2	0.3	0.4
SCC	best	0	96	96	96	89	94
	median	0	88	83	81	63	77
	worst	0	35	50	55	27	11
	mean	0	80	81	**81**	60	68
	std. dev.	0	16.3	13	12.8	22.7	25.4
IGD	best	0.1541	0.0009	0.0009	0.0009	0.0009	0.0009
	median	0.7393	0.0010	0.0010	0.0009	0.0010	0.0009
	worst	1.2865	0.0010	0.0010	0.0010	0.0010	0.0013
	mean	0.7591	0.0010	0.0010	**0.0009**	0.0010	0.0010
	std. dev.	0.3147	0.00003	0.00003	0.00003	0.00004	0.00009

Table C.2: Results obtained for functions ZDT3 and ZDT4, for sMOPSO, oMOPSO, and oMOPSO with fitness inheritance (p_i=0.1,0.2,0.3,0.4).

Test Function ZDT1						
SC(X,	sMOPSO)	oMOPSO)	0.1)	0.2)	0.3)	0.4)
sMOPSO	0.00	0.35	0.25	0.35	0.38	0.41
oMOPSO	0.08	0.00	0.22	**0.36**	**0.46**	**0.41**
0.1	0.08	**0.45**	0.00	0.46	0.51	0.52
0.2	0.06	0.29	0.24	0.00	0.38	0.40
0.3	0.05	0.21	0.16	0.28	0.00	0.32
0.4	0.05	0.28	0.16	0.31	0.36	0.00
Test Function ZDT2						
SC(X,	sMOPSO)	oMOPSO)	0.1)	0.2)	0.3)	0.4)
sMOPSO	0.00	0.00	0.00	0.00	0.00	0.00
oMOPSO	0.00	0.00	0.28	**0.34**	0.28	**0.47**
0.1	0.00	**0.31**	0.00	0.39	0.29	0.51
0.2	0.00	0.26	0.25	0.00	0.24	0.44
0.3	0.00	**0.35**	0.32	0.37	0.00	0.48
0.4	0.00	0.18	0.15	0.22	0.17	0.00
Test Function ZDT3						
SC(X,	sMOPSO)	oMOPSO)	0.1)	0.2)	0.3)	0.4)
sMOPSO	0.00	0.13	0.14	0.19	0.20	0.18
oMOPSO	0.23	0.00	0.28	**0.40**	**0.44**	**0.38**
0.1	0.25	**0.41**	0.00	0.49	0.53	0.45
0.2	0.22	0.26	0.22	0.00	0.38	0.31
0.3	0.22	0.20	0.17	0.29	0.00	0.26
0.4	0.23	0.26	0.22	0.34	0.39	0.00
Test Function ZDT4						
SC(X,	sMOPSO)	oMOPSO)	0.1)	0.2)	0.3)	0.4)
sMOPSO	0.00	0.00	0.00	0.00	0.00	0.00
oMOPSO	0.00	0.00	**0.38**	**0.32**	**0.54**	**0.42**
0.1	0.00	0.21	0.00	0.24	0.48	0.33
0.2	0.00	0.25	0.36	0.00	0.53	0.39
0.3	0.00	0.11	0.15	0.12	0.00	0.20
0.4	0.00	0.17	0.24	0.21	0.40	0.00

Table C.3: Results obtained for the test function ZDT4, for sMOPSO, oMOPSO, and oMOPSO with fitness inheritance (p_i=0.1,0.2,0.3,0.4).

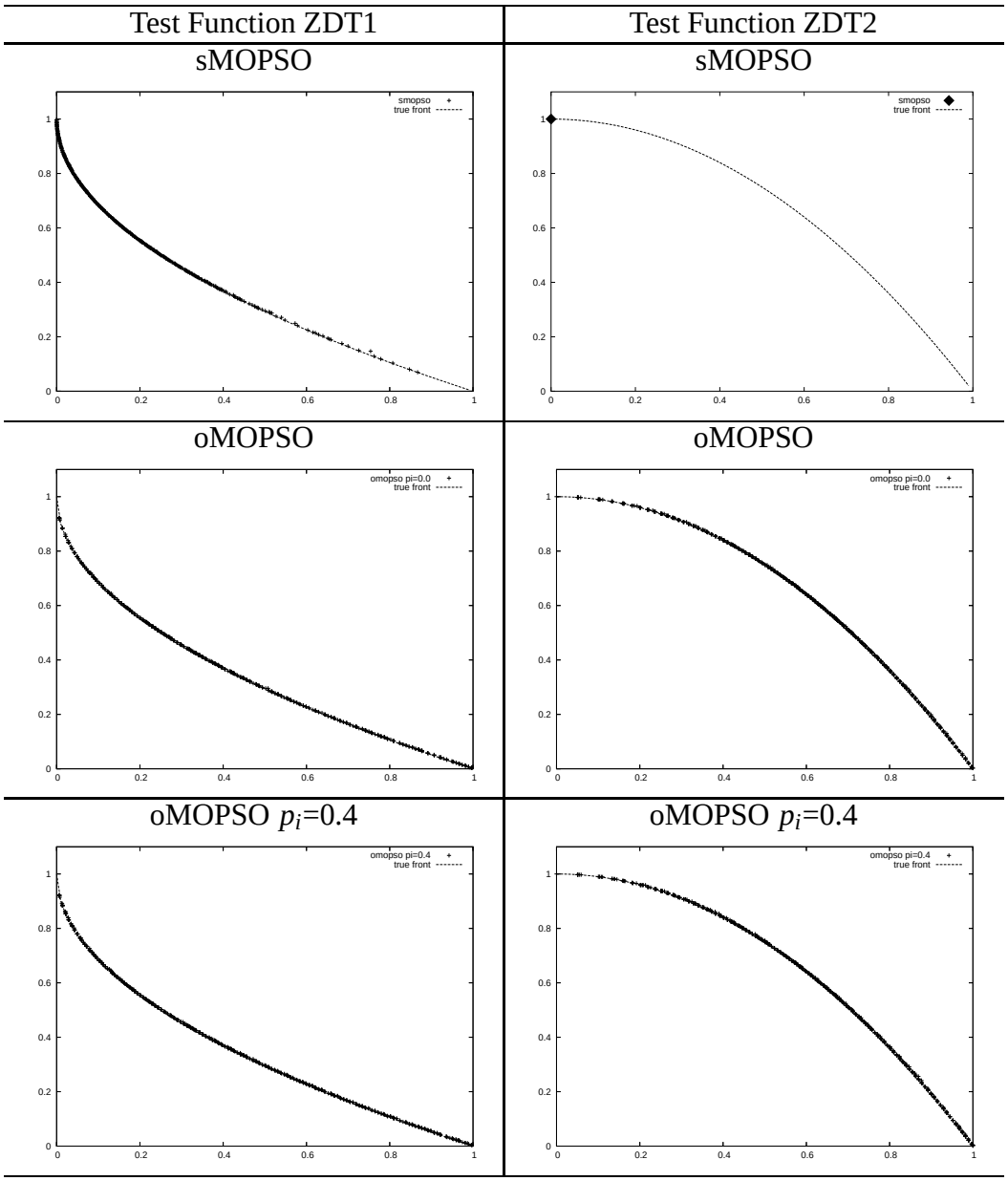


Figure C.1: Pareto fronts obtained by sMOPSO, oMOPSO, and oMOPSO with inheritance proportion of 0.4, for functions ZDT1 y ZDT2.

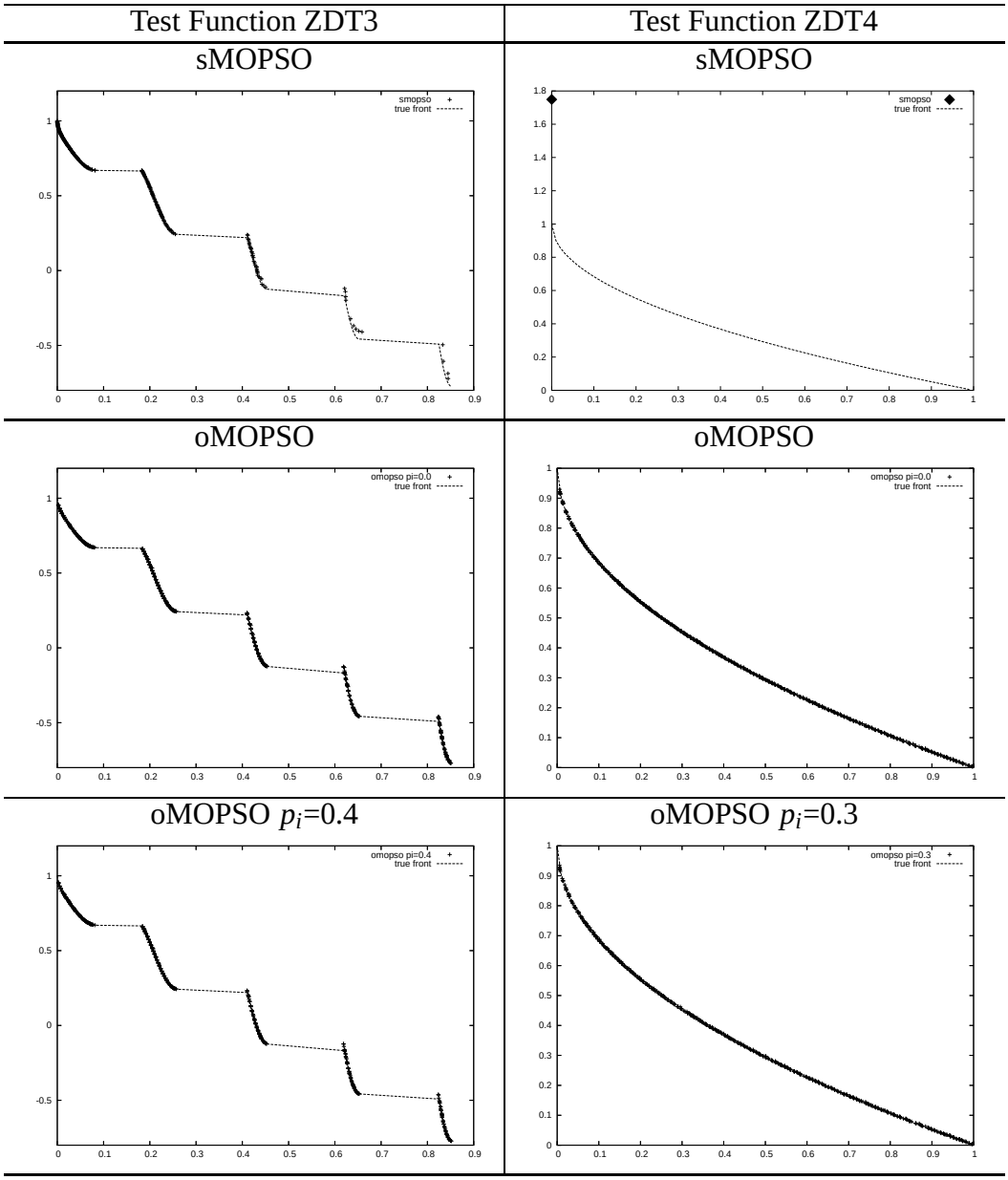


Figure C.2: Pareto fronts obtained by sMOPSO, oMOPSO, and oMOPSO with inheritance proportion of 0.4, for function ZDT3, and with inheritance proportion of 0.3, for function ZDT4.

FI1	Inheritance proportion p_i								
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)		0.4 (-40%)	
ZDT1	71	77	(+8.5%)	64	(-9.9%)	62	(-12.7%)	61	(-14.1%)
ZDT2	89	83	(-6.7%)	86	(-3.4%)	79	(-11.2%)	77	(-13.5%)
ZDT3	68	73	(+7.4%)	65	(-4.4%)	64	(-5.9%)	59	(-13.2%)
ZDT4	80	81	(+1.3%)	81	(+1.3%)	60	(-25.0%)	68	(-15.0%)
Average			**+2.6%**		**-4.1 %**		**-13.7 %**		**-14.0 %**
FI2	Inheritance proportion p_i								
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)		0.4 (-40%)	
ZDT1	71	74	(+4.2%)	68	(-4.2%)	68	(-4.2%)	59	(-16.9%)
ZDT2	89	81	(-9.0%)	82	(-7.9%)	78	(-12.4%)	77	(-13.5%)
ZDT3	68	64	(-5.9%)	67	(-1.5%)	58	(-14.7%)	63	(-7.4%)
ZDT4	80	77	(-3.8%)	83	(+3.8%)	67	(-16.3%)	69	(-13.8%)
Average			**-3.6%**		**-2.4 %**		**-11.9 %**		**-12.9 %**
FI3	Inheritance proportion p_i								
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)		0.4 (-40%)	
ZDT1	71	73	(+2.8%)	69	(-2.8%)	69	(-2.8%)	50	(-29.6%)
ZDT2	89	87	(-2.2%)	82	(-7.9%)	71	(-20.2%)	76	(-14.6%)
ZDT3	68	67	(-1.5%)	63	(-7.4%)	64	(-5.9%)	60	(-11.8%)
ZDT4	80	81	(+1.3%)	79	(-1.3%)	59	(-26.3%)	68	(-15.0%)
Average			**+0.1%**		**-4.9 %**		**-13.8 %**		**-17.8%**
FI4	Inheritance proportion p_i								
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)		0.4 (-40%)	
ZDT1	71	62	(-12.7%)	62	(-12.7%)	59	(-16.9%)	49	(-31.0%)
ZDT2	89	85	(-4.5%)	84	(-5.6%)	78	(-12.4%)	79	(-11.2%)
ZDT3	68	73	(+7.4%)	69	(+1.5%)	60	(-11.8%)	58	(-14.7%)
ZDT4	80	88	(+10.0%)	88	(+10.0%)	85	(+6.3%)	82	(+2.5%)
Average			**+0.1%**		**-1.7 %**		**-8.7 %**		**-13.6%**
FI5	Inheritance proportion p_i								
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)		0.4 (-40%)	
ZDT1	71	74	(+4.2%)	69	(-2.8%)	61	(-14.1%)	56	(-21.1%)
ZDT2	89	89	(0.0%)	79	(-11.2%)	84	(-5.6%)	77	(-13.5%)
ZDT3	68	72	(+5.9%)	70	(+2.9%)	55	(-19.1%)	58	(-14.7%)
ZDT4	80	87	(+8.8%)	85	(+6.3%)	85	(+6.3%)	82	(+2.5%)
Average			**+4.7%**		**-1.2 %**		**-8.1 %**		**-11.7%**

Table C.4: Results obtained for different values of inheritance proportion, for techniques FI1, FI2, FI3, FI4 and FI5.

FI6	Inheritance proportion p_i								
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)		0.4 (-40%)	
ZDT1	71	70	(-1.4%)	61	(-14.1%)	62	(-12.7%)	47	(-33.8%)
ZDT2	89	83	(-6.7%)	82	(-7.9%)	76	(-14.6%)	70	(-21.3%)
ZDT3	68	72	(+5.9%)	72	(+5.9%)	59	(-13.2%)	61	(-10.3%)
ZDT4	80	83	(+3.8%)	84	(+5.0%)	80	(0.0%)	79	(-1.3%)
Average			**+1.6%**		**-2.8 %**		**-10.1 %**		**-16.7%**
FI7	Inheritance proportion p_i								
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)		0.4 (-40%)	
ZDT1	71	64	(-9.9%)	58	(-18.3%)	57	(-19.7%)	47	(-33.8%)
ZDT2	89	83	(-6.7%)	74	(-16.9%)	68	(-23.6%)	66	(-25.8%)
ZDT3	68	66	(-2.9%)	69	(+1.5%)	64	(-5.9%)	57	(-16.2%)
ZDT4	80	80	(0.0%)	74	(-7.5%)	57	(-28.8%)	44	(-45.0%)
Average			**-4.9%**		**-10.3 %**		**-19.5 %**		**-30.2%**
FI8	Inheritance proportion p_i								
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)		0.4 (-40%)	
ZDT1	71	69	(-2.8%)	62	(-12.7%)	53	(-25.4%)	47	(-33.8%)
ZDT2	89	85	(-4.5%)	84	(-5.6%)	66	(-25.8%)	65	(-27.0%)
ZDT3	68	71	(+4.4%)	67	(-1.5%)	61	(-10.3%)	52	(-23.5%)
ZDT4	80	80	(0.0%)	72	(-10.0%)	63	(-21.3%)	52	(-35.0%)
Average			**-0.7%**		**-7.5 %**		**-20.7 %**		**-29.8%**
FI9	Inheritance proportion p_i								
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)		0.4 (-40%)	
ZDT1	71	67	(-5.6%)	58	(-18.3%)	54	(-23.9%)	44	(-38.0%)
ZDT2	89	90	(+1.1%)	85	(-4.5%)	69	(-22.5%)	68	(-23.6%)
ZDT3	68	68	(0.0%)	67	(-1.5%)	61	(-10.3%)	51	(-25.0%)
ZDT4	80	83	(+3.8%)	76	(-5.0%)	72	(-10.0%)	58	(-27.5%)
Average			**-0.2%**		**-7.3 %**		**-16.7 %**		**-28.5%**
FI10	Inheritance proportion p_i								
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)		0.4 (-40%)	
ZDT1	71	71	(0.0%)	58	(-18.3%)	59	(-16.9%)	48	(-32.4%)
ZDT2	89	78	(-12.4%)	78	(-12.4%)	69	(-22.5%)	58	(-34.8%)
ZDT3	68	70	(+2.9%)	63	(-7.4%)	61	(-10.3%)	47	(-30.9%)
ZDT4	80	78	(-2.5%)	81	(+1.3%)	58	(-27.5%)	52	(-35.0%)
Average			**-3.0%**		**-9.2 %**		**-19.3 %**		**-33.3%**

Table C.5: Results obtained for different values of inheritance proportion, for techniques FI6, FI7, FI8, FI9 and FI10.

FI11	Inheritance proportion p_i								
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)		0.4 (-40%)	
ZDT1	71	62	(-12.7%)	63	(-11.3%)	55	(-22.5%)	37	(-48.0%)
ZDT2	89	84	(-5.6%)	87	(-2.2%)	81	(-9.0%)	76	(-14.6%)
ZDT3	68	69	(+1.5%)	60	(-11.8%)	57	(-16.2%)	44	(-35.3%)
ZDT4	80	82	(+2.5%)	81	(+1.3%)	73	(-8.8%)	73	(-8.8%)
Average			**-3.6%**		**-6.0 %**		**-14.1 %**		**-26.7%**
FI12	Inheritance proportion p_i								
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)		0.4 (-40%)	
ZDT1	71	66	(-7.0%)	56	(-21.1%)	55	(-22.5%)	48	(-32.4%)
ZDT2	89	87	(-2.2%)	85	(-4.5%)	74	(-16.9%)	80	(-10.1%)
ZDT3	68	66	(-2.9%)	64	(-5.9%)	55	(-19.1%)	53	(-22.1%)
ZDT4	80	80	(0.0%)	75	(-6.3%)	71	(-11.3%)	61	(-23.8%)
Average			**-3.0%**		**-9.5 %**		**-17.5 %**		**-22.1%**
FI13	Inheritance proportion p_i								
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)		0.4 (-40%)	
ZDT1	71	68	(-4.2%)	69	(-2.8%)	63	(-11.3%)	58	(-18.3%)
ZDT2	89	84	(-5.6%)	81	(-9.0%)	79	(-11.2%)	80	(-10.1%)
ZDT3	68	70	(+2.9%)	68	(0.0%)	63	(-7.4%)	54	(-20.6%)
ZDT4	80	79	(-1.3%)	81	(+1.3%)	64	(-20.0%)	59	(-26.3%)
Average			**-2.1%**		**-2.6 %**		**-12.5 %**		**-18.8%**
FI14	Inheritance proportion p_i								
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)		0.4 (-40%)	
ZDT1	71	75	(+5.6%)	66	(-7.0%)	58	(-18.3%)	59	(-16.9%)
ZDT2	89	88	(-1.1%)	79	(-11.2%)	83	(-6.7%)	72	(-19.1%)
ZDT3	68	74	(+8.8%)	69	(+1.5%)	63	(-7.4%)	60	(-11.8%)
ZDT4	80	81	(+1.3%)	79	(-1.3%)	73	(-8.8%)	67	(-16.3%)
Average			**+3.7%**		**-4.9 %**		**-10.3 %**		**-16.0%**
FI15	Inheritance proportion p_i								
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)		0.4 (-40%)	
ZDT1	71	69	(-2.8%)	63	(-11.3%)	69	(-2.8%)	56	(-21.1%)
ZDT2	89	86	(-3.4%)	81	(-9.0%)	72	(-19.1%)	73	(-18.0%)
ZDT3	68	72	(+5.9%)	70	(+2.9%)	64	(-5.9%)	58	(-14.7%)
ZDT4	80	81	(+1.3%)	78	(-2.5%)	63	(-21.3%)	70	(-12.5%)
Average			**+0.3%**		**-5.0 %**		**-12.3 %**		**-16.6%**

Table C.6: Results obtained for different values of inheritance proportion, for techniques FI11, FI12, FI13, FI14 and FI15.

FA1	Approximation proportion p_a							
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)	0.4 (-40%)	
ZDT1	71	74	(+4.2%)	64	(-9.9%)	63	(-11.3%)	55	(-22.5%)
ZDT2	89	88	(-1.1%)	85	(-4.5%)	81	(-9.0%)	76	(-14.6%)
ZDT3	68	73	(+7.4%)	61	(-10.3%)	60	(-11.8%)	55	(-19.1%)
ZDT4	80	85	(+6.3%)	89	(+11.3%)	79	(-1.3%)	80	(0.0%)
Average			**+4.2%**		**-3.4 %**		**-8.4 %**		**-14.1%**
FA2	Approximation proportion p_a							
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)	0.4 (-40%)	
ZDT1	71	75	(+5.6%)	57	(-19.7%)	54	(-23.9%)	46	(-35.2%)
ZDT2	89	83	(-6.7%)	72	(-19.1%)	63	(-29.2%)	76	(-14.6%)
ZDT3	68	63	(-7.4%)	58	(-14.7%)	58	(-14.7%)	56	(-17.6%)
ZDT4	80	86	(+7.5%)	87	(+8.8%)	81	(+1.3%)	83	(+3.8%)
Average			**-0.3%**		**-11.2 %**		**-16.6 %**		**-15.9%**
FA3	Approximation proportion p_a							
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)	0.4 (-40%)	
ZDT1	71	71	(0.0%)	67	(-5.6%)	63	(-11.3%)	50	(-29.6%)
ZDT2	89	88	(-1.1%)	87	(-2.2%)	85	(-4.5%)	76	(-14.6%)
ZDT3	68	65	(-4.4%)	65	(-4.4%)	55	(-19.1%)	57	(-16.2%)
ZDT4	80	89	(+11.3%)	91	(+13.8%)	86	(+7.5%)	87	(+8.8%)
Average			**+1.5%**		**+0.4 %**		**-6.9 %**		**-12.9%**
FA4	Approximation proportion p_a							
function	0.0	0.1 (-10%)		0.2 (-20%)		0.3 (-30%)	0.4 (-40%)	
ZDT1	71	69	(-2.8%)	59	(-16.9%)	60	(-15.5%)	52	(-26.8%)
ZDT2	89	87	(-2.2%)	80	(-10.1%)	76	(-14.6%)	71	(-20.2%)
ZDT3	68	67	(-1.5%)	71	(+4.4%)	56	(-17.6%)	56	(-17.6%)
ZDT4	80	86	(+7.5%)	85	(+6.3%)	79	(-1.3%)	80	(0.0%)
Average			**+0.3%**		**-4.1 %**		**-12.3 %**		**-16.2%**

Table C.7: Results obtained for different values of approximation proportion, for techniques FA1, FA2, FA3 and FA4.

Test Function DTLZ2								
		sMOPSO	cMOPSO	oMOPSO	0.1	0.2	0.3	0.4
SCC	mean	**25**	16	18	18	12	13	12
	std. dev.	4.2	6.7	6.9	7.7	5.2	6.4	6.8
IGD	mean	0.0014	0.0021	**0.0014**	0.0014	0.0015	0.0015	0.0015
	std. dev.	0.00005	0.0004	0.00004	0.00005	0.0001	0.00008	0.00009
Test Function DTLZ6								
		sMOPSO	cMOPSO	oMOPSO	0.1	0.2	0.3	0.4
SCC	mean	1	0	**62**	61	60	61	43
	std. dev.	0	0	13	17.3	21.7	17.2	20.7
IGD	mean	0.0673	0.0373	0.0091	**0.0074**	0.0089	0.0087	0.0101
	std. dev.	0.0000	0.0172	0.0058	0.0060	0.0060	0.0058	0.0058

Table C.8: Results obtained for the test functions DTLZ2 and DTLZ6, for sMOPSO, cMOPSO, oMOPSO, and oMOPSO with the fitness inheritance technique FI5 incorporated (p_i=0.1,0.2,0.3,0.4).

Test Function DTLZ2								
		sMOPSO	cMOPSO	oMOPSO	0.1	0.2	0.3	0.4
SCC	mean	**25**	16	18	16	15	16	13
	std. dev.	4.2	6.7	6.9	6	7.7	8.2	7.3
IGD	mean	0.0014	0.0021	**0.0014**	0.0014	0.0015	0.0015	0.0015
	std. dev.	0.00005	0.0004	0.00004	0.00005	0.00007	0.0001	0.0001
Test Function DTLZ6								
		sMOPSO	cMOPSO	oMOPSO	0.1	0.2	0.3	0.4
SCC	mean	1	0	**62**	62	51	54	53
	std. dev.	0	0	13	15	27	20	20
IGD	mean	0.0673	0.0373	**0.0091**	0.0109	0.0099	0.0100	0.0116
	std. dev.	0.0000	0.0172	0.0058	0.0056	0.0059	0.0061	0.0056

Table C.9: Results obtained for the test functions DTLZ2 and DTLZ6, for sMOPSO, cMOPSO, oMOPSO, and oMOPSO with the fitness approximation technique FA3 incorporated (p_a=0.1,0.2,0.3,0.4).

Function ZDT1								
		noinherit	nlinear1	nlinear2	nlinear3	linear	nlinear4	nlinear5
SCC	best	99	99	94	97	95	81	62
	median	93	87	79	76	75	55	17
	worst	47	38	16	29	12	10	1
	mean	**87**	**84**	**74**	**71**	**68**	**53**	**21**
	st. dev.	12.5	12.6	21	18.6	22.7	21.6	13.5
IGD	best	0.0009	0.0009	0.0009	0.0009	0.0009	0.0009	0.0010
	median	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0014
	worst	0.0010	0.0010	0.0020	0.0492	0.0318	0.0492	0.0831
	mean	**0.0010**	**0.0010**	**0.0010**	**0.0031**	**0.0028**	**0.0039**	**0.0084**
	st. dev.	0.0000	0.0000	0.0002	0.0089	0.0069	0.0098	0.0180
evaluations		20200	16306	13640	10295	10303	6966	4319
savings		0%	19.3%	32.5%	49%	49%	65.5%	78.6%

Function ZDT2								
		noinherit	nlinear1	nlinear2	nlinear3	linear	nlinear4	nlinear5
SCC	best	100	100	100	99	100	98	95
	median	96	94	94	91	93	79	46
	worst	35	69	47	0	2	0	0
	mean	**92**	**93**	**89**	**83**	**84**	**69**	**45**
	st. dev.	12.9	6.1	12.2	21.7	22.9	26.6	34.2
IGD	best	0.0006	0.0006	0.0006	0.0006	0.0006	0.0006	0.0007
	median	0.0007	0.0007	0.0007	0.0007	0.0007	0.0007	0.0011
	worst	0.0010	0.0008	0.0008	0.0059	0.0035	0.0678	0.0456
	mean	**0.0007**	**0.0007**	**0.0007**	**0.0009**	**0.0008**	**0.0052**	**0.0038**
	st. dev.	0.0001	0.0000	0.0000	0.0010	0.0005	0.0139	0.0090
evaluations		20200	16304	13641	10295	10298	6968	4316
savings		0%	19.3%	32.5%	49%	49%	65.5%	78.6%

Table C.10: Results obtained for all the test functions and all the adaptive functions.

Function ZDT3								
		noinherit	nlinear1	nlinear2	nlinear3	linear	nlinear4	nlinear5
SCC	best	91	91	93	86	89	69	48
	median	78	74	74	57	60	37	11
	worst	42	38	9	4	17	10	2
	mean	**76**	**73**	**72**	**53**	**59**	**37**	**16**
	st. dev.	12.7	11.6	15.9	21.5	16.2	18	12.6
IGD	best	0.0007	0.0008	0.0008	0.0009	0.0008	0.0009	0.0017
	median	0.0008	0.0009	0.0009	0.0019	0.0011	0.0028	0.0129
	worst	0.0013	0.0020	0.0159	0.0542	0.0273	0.0491	0.0505
	mean	**0.0009**	**0.0010**	**0.0020**	**0.0074**	**0.0023**	**0.0109**	**0.0178**
	st. dev.	0.0001	0.0003	0.0032	0.0138	0.0049	0.0137	0.0147
evaluations		20200	16312	13622	10290	10304	6966	4336
savings		0%	19.2%	32.6%	49.1%	49%	65.5%	78.5%

Function ZDT4								
		noinherit	nlinear1	nlinear2	nlinear3	linear	nlinear4	nlinear5
SCC	best	100	99	98	99	99	95	81
	median	97	97	96	94	95	83	48
	worst	78	69	74	49	28	3	2
	mean	**96**	**94**	**93**	**89**	**90**	**77**	**47**
	st. dev.	4.8	6.6	6.0	12.6	14.2	18.1	22.6
IGD	best	0.0009	0.0009	0.0009	0.0009	0.0009	0.0009	0.0010
	median	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0011
	worst	0.0010	0.0010	0.0010	0.0010	0.0010	0.0014	0.0021
	mean	**0.0010**	**0.0010**	**0.0010**	**0.0010**	**0.0010**	**0.0010**	**0.0012**
	st. dev.	0.0000	0.0000	0.0000	0.0000	0.0000	0.0001	0.0003
evaluations		20200	16287	13626	10315	10304	6958	4315
savings		0%	19.4%	32.5%	48.9%	49%	65.6%	78.6%

Table C.11: Results obtained for all the test functions and all the adaptive functions.

Function DTLZ6								
		noinherit	nlinear1	nlinear2	nlinear3	linear	nlinear4	nlinear5
SCC	best	97	100	93	75	84	73	38
	median	74	68	67	52	51	41	21
	worst	42	41	37	21	16	16	1
	mean	**74**	**71**	**66**	**52**	**52**	**42**	**20**
	st. dev.	14.2	15.8	14.2	14.6	19.2	14.3	7.9
IGD	best	0.0023	0.0023	0.0026	0.0030	0.0026	0.0031	0.0095
	median	0.0030	0.0136	0.0138	0.0141	0.0140	0.0147	0.0210
	worst	0.0187	0.0166	0.0165	0.0205	0.0248	0.0231	0.0447
	mean	**0.0064**	**0.0087**	**0.0104**	**0.0126**	**0.0118**	**0.0139**	**0.0205**
	st. dev.	0.0053	0.0057	0.0053	0.0048	0.0067	0.0061	0.0082
evaluations		20200	16303	13643	10297	10304	6959	4343
savings		0%	19.3%	32.5%	49%	50.9%	65.5%	78.5%

Table C.12: Results obtained for all the test functions and all the adaptive functions.

Confidence Intervals for the mean of SCC					
approach	ZDT1	ZDT2	ZDT3	ZDT4	DTLZ6
no-inherit	[82.1,91.5]	[87.0,95.4]	[71.3,80.8]	[94.7,96.8]	[67.8,78.1]
nonlinear1	[79.1,88.6]	[91.1,95.6]	[69.1,77.7]	[91.2,95.9]	[62.3,74.5]
nonlinear2	[65.5,80.2]	[82.3,93.0]	[66.1,78.0]	[91.1,94.9]	[60.2,70.0]
nonlinear3	[63.6,77.7]	[72.3,87.0]	[45.3,61.4]	[85.0,92.4]	[46.3,57.1]
linear	[60.0,76.9]	[76.2,90.4]	[52.5,64.6]	[85.2,93.8]	[43.2,56.7]
nonlinear4	[45.2,61.3]	[58.6,78.5]	[30.3,43.8]	[69.2,81.6]	[36.4,46.2]
nonlinear5	[15.6,25.7]	[31.7,57.3]	[11.8,21.2]	[38.2,55.1]	[16.9,22.3]

Table C.13: Confidence intervals for the mean of the Success Counting measure.

Confidence Intervals for the mean of IGD			
approach	function ZDT1	function ZDT2	function ZDT3
no-inherit	[0.000945,0.000964]	[0.000647,0.000686]	[0.000874,0.000962]
nonlinear1	[0.000945,0.000971]	[0.000647,0.000666]	[0.000954,0.001130]
nonlinear2	[0.000946,0.001125]	[0.000657,0.000688]	[0.001198,0.003866]
nonlinear3	[0.001807,0.004922]	[0.000676,0.001372]	[0.003199,0.015646]
linear	[0.000970,0.006952]	[0.000674,0.001084]	[0.001197,0.004792]
nonlinear4	[0.001537,0.009697]	[0.001544,0.011302]	[0.007711,0.015563]
nonlinear5	[0.005391,0.019967]	[0.001469,0.010545]	[0.012888,0.023290]

Table C.14: Confidence intervals for the mean of the Inverted Generational Distance measure.

Confidence Intervals for the mean of IGD		
approach	function ZDT4	function DTLZ6
no-inherit	[0.000943,0.000966]	[0.004636,0.007801]
nonlinear1	[0.000950,0.000967]	[0.007273,0.011590]
nonlinear2	[0.000949,0.000967]	[0.008654,0.011922]
nonlinear3	[0.000939,0.000961]	[0.010853,0.014540]
linear	[0.000946,0.000971]	[0.009773,0.014772]
nonlinear4	[0.000964,0.001030]	[0.011890,0.016370]
nonlinear5	[0.001122,0.001359]	[0.017997,0.024043]

Table C.15: Confidence intervals for the mean of the Inverted Generational Distance measure.

Function ZDT1						
X	nl1	nl2	nl3	lin	nl4	nl5
SC(no-inh,X)	0.33	0.23	0.25	0.25	0.23	0.66
SC(X,no-inh)	0.04	0.02	0.00	0.00	0.00	0.00
Function ZDT2						
X	nl1	nl2	nl3	lin	nl4	nl5
SC(no-inh,X)	0.37	0.21	0.29	0.20	0.39	0.30
SC(X,no-inh)	0.03	0.07	0.05	0.04	0.00	0.00
Function ZDT3						
X	nl1	nl2	nl3	lin	nl4	nl5
SC(no-inh,X)	0.27	0.20	0.34	0.37	0.40	0.67
SC(X,no-inh)	0.14	0.08	0.02	0.01	0.00	0.00
Function ZDT4						
X	nl1	nl2	nl3	lin	nl4	nl5
SC(no-inh,X)	0.21	0.12	0.19	0.13	0.11	0.30
SC(X,no-inh)	0.05	0.14	0.06	0.01	0.01	0.01
Function DTLZ6						
X	nl1	nl2	nl3	lin	nl4	nl5
SC(no-inh,X)	0.01	0.05	0.09	0.10	0.12	0.13
SC(X,no-inh)	0.01	0.01	0.01	0.01	0.00	0.00

Table C.16: Set Coverage measure.

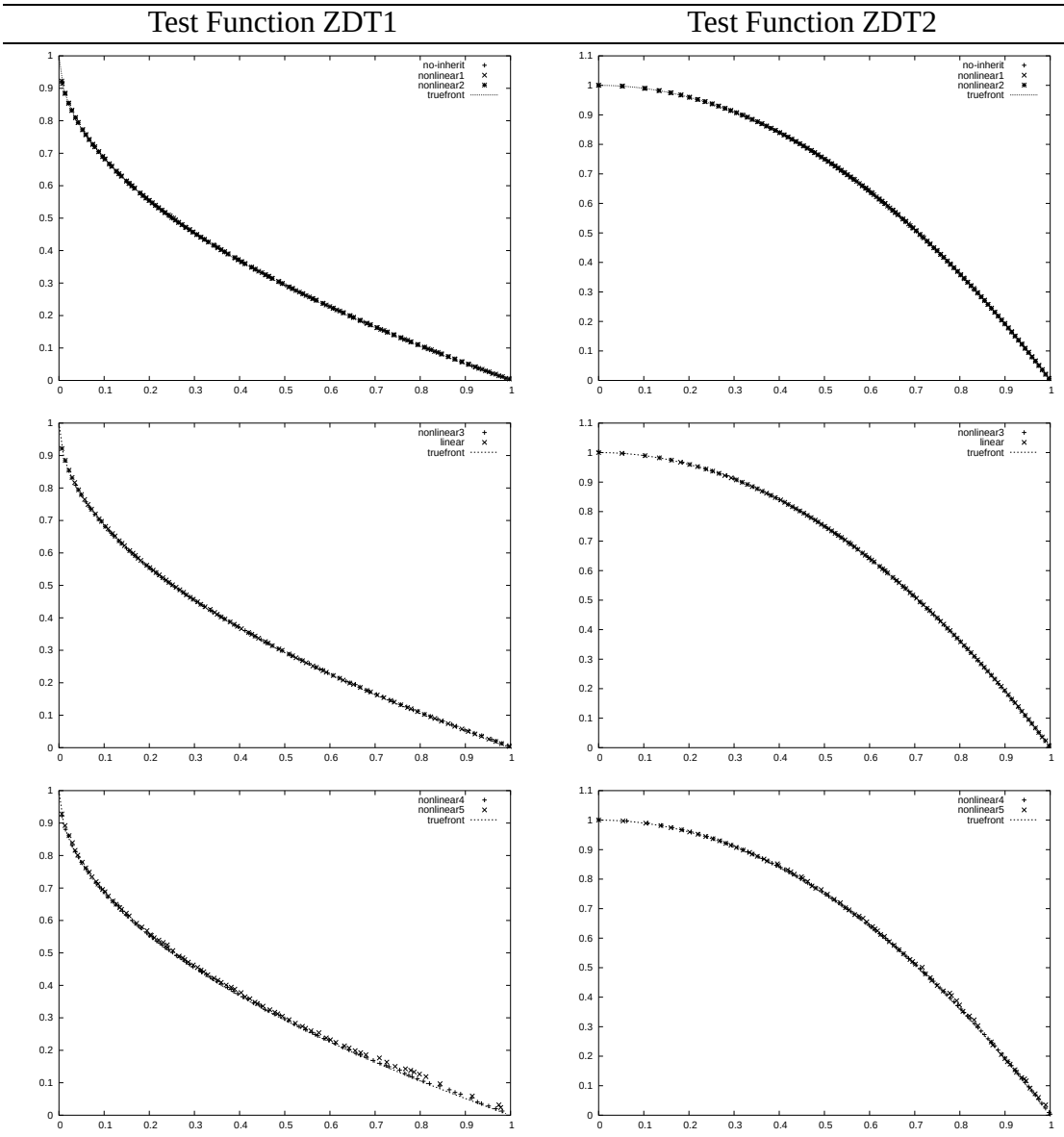


Figure C.3: Pareto fronts obtained for functions ZDT1 and ZDT2.

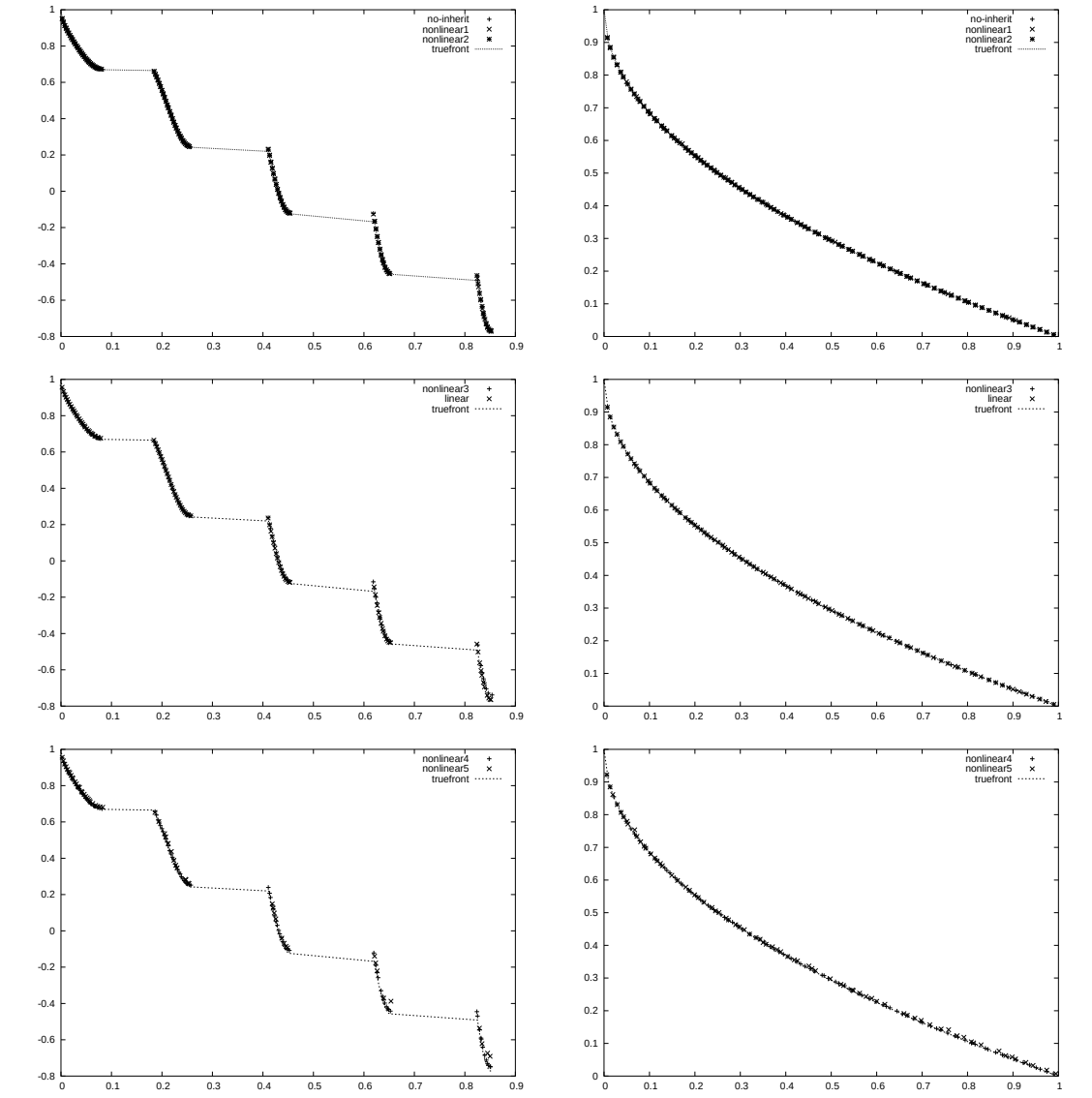


Figure C.4: Pareto fronts obtained for functions ZDT3 and ZDT4.

Test Function DTLZ6

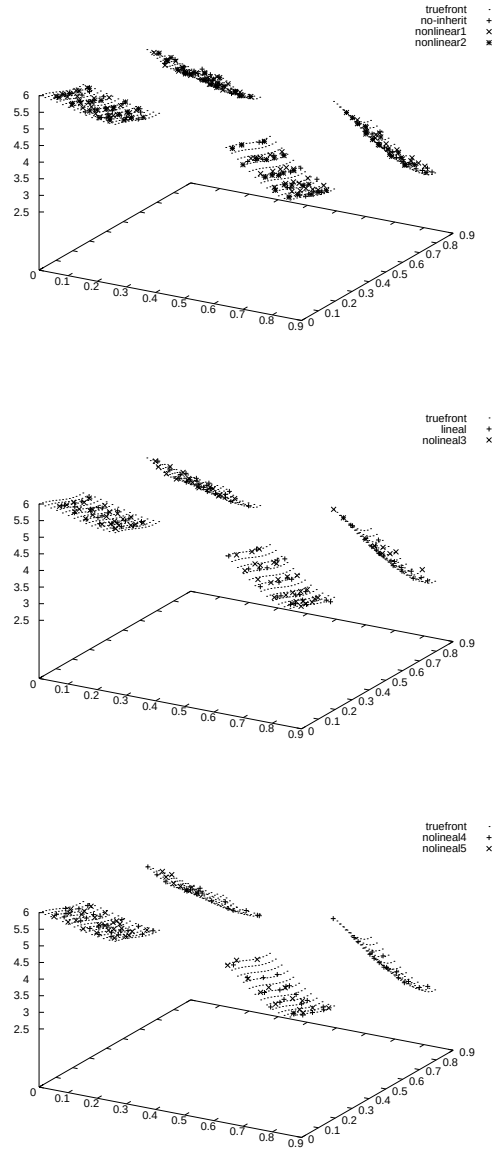


Figure C.5: Pareto fronts obtained for function DTLZ6.

References

[1] Julio E. Alvarez-Benitez, Richard M. Everson, and Jonathan E. Fieldsend. A MOPSO Algorithm Based Exclusively on Pareto Dominance Concepts. In *Third International Conference on Evolutionary Multi-Criterion Optimization, EMO 2005.*, pages 459–473, Guanajuato, México, 2005. LNCS 3410, Springer-Verlag.

[2] Thomas Bäck, editor. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

[3] Richard Balling. The Maximin Fitness Function; Multiobjective City and Regional Planning. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, pages 1–15, Faro, Portugal, April 2003. Springer. Lecture Notes in Computer Science. Volume 2632.

[4] Helio J. C. Barbosa. A coevolutionary genetic algorithm for a game approach to structural optimization. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 545–552, San Mateo, California, July 1997. Michigan State University, Morgan Kaufmann Publishers.

[5] Helio J.C. Barbosa and André M.S. Barreto. An interactive genetic algorithm with co-evolution of weights for multiobjective problems. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 203–210, San Francisco, California, 2001. Morgan Kaufmann Publishers.

[6] Thomas Bartz-Beielstein, Philipp Limbourg, Konstantinos E. Parsopoulos, Michael N. Vrahatis, Jörn Mehnen, and Karlheinz Schmitt. Particle Swarm Optimizers for Pareto Optimization with Enhanced Archiving Techniques. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC'2003)*, volume 3, pages 1780–1787, Canberra, Australia, December 2003. IEEE Press.

[7] U. Baumgartner, Ch. Magele, and W. Renhart. Pareto Optimality and Particle Swarm Optimization. *IEEE Transactions on Magnetics*, 40(2):1172–1175, March 2004.

[8] J. Branke and C. Schmidt. Faster convergence by means of fitness estimation. *Soft Computing*, 9(1):13–20, January 2005.

[9] A. J. Brooker, J. Dennis, P. D. Frank, D. B. Serafini, V. Torczon, and M. Trosset. A Rigorous Framework for Optimization of Expensive Functions by Surrogates. *Structural Optimization*, 9:1–13, 1998.

[10] Lam T. Bui, Hussein A. Abbass, and Daryl Essam. Fitness inheritance for noisy evolutionary multi-objective optimization. In *Proc. of the Genetic and Evolutionary Computation Conference*, pages 25–29. ACM, 2005.

[11] Jian-Jung Chen. *Theory and Applications of Efficient Multi-Objective Evolutionary Algorithms*. PhD thesis, Feng Chia University, Taichung, Taiwan, 2004.

[12] Jian-Jung Chen, David E. Goldberg, Shinn-Ying Ho, and Kumara Sastry. Fitness Inheritance in Multi-Objective Optimization. In W.B. Langdon et.al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2002)*, pages 319–326, San Francisco, California, July 2002. Morgan Kaufmann Publishers.

[13] Stephen L. Chiu. Fuzzy model identification based on cluster estimation. *Journal of Intelligent and Fuzzy Systems*, 2(3):267–278, September 1994.

[14] Chi-kin Chow and Hung-tat Tsui. Autonomous Agent Response Learning by a Multi-Species Particle Swarm Optimization. In *2004 Congress on Evolutionary Computation (CEC'2004)*, volume 1, pages 778–785, Portland, Oregon, USA, June 2004. IEEE Service Center.

[15] Maurice Clerc and James Kennedy. The particle swarm–explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, February 2002.

[16] Carlos A. Coello Coello and Margarita Reyes Sierra. A Coevolutionary Multi-Objective Evolutionary Algorithm. In *Proceedings of 2003 Congress on Evolutionary Computation*, volume 1, pages 482–489, Canberra, Australia, December 2003. IEEE Press.

[17] Carlos A. Coello Coello and Maximino Salazar Lechuga. MOPSO: A Proposal for Multiple Objective Particle Swarm Optimization. In *Congress on Evolutionary Computation (CEC'2002)*, volume 2, pages 1051–1056, Piscataway, New Jersey, May 2002. IEEE Service Center.

[18] Carlos A. Coello Coello and Gregorio Toscano Pulido. A Micro-Genetic Algorithm for Multiobjective Optimization. In Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, and David Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 126–140. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

[19] Carlos A. Coello Coello and Gregorio Toscano Pulido. Multiobjective Optimization using a Micro-Genetic Algorithm. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 274–282, San Francisco, California, 2001. Morgan Kaufmann Publishers.

[20] Carlos A. Coello Coello, Gregorio Toscano Pulido, and Maximino Salazar Lechuga. Handling Multiple Objectives With Particle Swarm Optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):256–279, June 2004.

[21] Carlos A. Coello Coello, David A. Van Veldhuizen, and Gary B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, May 2002.

[22] J. L. Cohon and D. H. Marks. A Review and Evaluation of Multiobjective Programming Techniques. *Water Resources Research*, 11(2):208–220, 1975.

[23] David W. Corne, Nick R. Jerram, Joshua D. Knowles, and Martin J. Oates. PESA-II: Region-based selection in evolutionary multiobjective optimization. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 283–290, San Francisco, California, 2001. Morgan Kaufmann Publishers.

[24] David W. Corne, Joshua D. Knowles, and Martin J. Oates. The pareto envelope-based selection algorithm for multiobjective optimization. In Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo, and Hans-Paul Schwefel, editors, *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 839–848, Paris, France, 2000. Springer. Lecture Notes in Computer Science No. 1917.

[25] Kalyanmoy Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, Chichester, UK, 2001. ISBN 0-471-87339-X.

[26] Kalyanmoy Deb, Samir Agrawal, Amrit Pratab, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo, and Hans-Paul Schwefel, editors, *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 849–858, Paris, France, 2000. Springer. Lecture Notes in Computer Science No. 1917.

[27] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.

[28] Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. Scalable Multi-Objective Optimization Test Problems. In *Congress on Evolutionary Computation (CEC'2002)*, volume 1, pages 825–830, Piscataway, New Jersey, May 2002. IEEE Service Center.

[29] Els I. Ducheyne, Bernard De Baets, and Robert De Wulf. Is Fitness Inheritance Useful for Real-World Applications? In C. M. Fonseca et.al., editor,

Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003, pages 31–42, Faro, Portugal, April 2003. Springer. Lecture Notes in Computer Science. Volume 2632.

[30] P.R. Ehrlich and P.H. Raven. Butterflies and Plants: A Study in Coevolution. *Evolution*, 18:586–608, 1964.

[31] Andries P. Engelbrecht, editor. *Computational Intelligence: An Introduction*. John Wiley & Sons, England, 2002.

[32] Andries P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. John Wiley & Sons, 2005.

[33] Mark Erickson, Alex Mayer, and Jeffrey Horn. The Niched Pareto Genetic Algorithm 2 applied to the design of groundwater remediation systems. In Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, and David Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 681–695. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

[34] Jonathan E. Fieldsend and Sameer Singh. A Multi-Objective Algorithm based upon Particle Swarm Optimisation, an Efficient Data Structure and Turbulence. In *Proceedings of the 2002 U.K. Workshop on Computational Intelligence*, pages 37–44, Birmingham, UK, September 2002.

[35] Lawrence J. Fogel. *Artificial Intelligence through Simulated Evolution*. John Wiley, New York, 1966.

[36] Carlos M. Fonseca and Peter J. Fleming. Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423, San Mateo, California, 1993. University of Illinois at Urbana-Champaign, Morgan Kauffman Publishers.

[37] M. P. Fourman. Compaction of symbolic layout using genetic algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 141–153. Lawrence Erlbaum, 1985.

[38] A. S. Fraser. Simulation of genetic systems by automatic digital computers. *Australian Journal of Biological Sciences*, 10:484–499, 1957.

[39] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Co., Reading, Massachusetts, 1989.

[40] David E. Goldberg and Kalyanmoy Deb. A comparison of selection schemes used in genetic algorithms. In Gregory J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, San Mateo, California, 1991.

[41] David E. Goldberg and Jon Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49. Lawrence Erlbaum Associates, 1987.

[42] P. Hajela and C. Y. Lin. Genetic search strategies in multicriterion optimal design. *Structural Optimization*, 4:99–107, 1992.

[43] K. Hirasawa, J. Ishikawa, J. Hu, C. Jin, and J. Murata. Genetic Symbiosis Algorithm. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1377–1384. IEEE Press, July 2000.

[44] S.L. Ho, Yang Shiyou, Ni Guangzheng, Edward W.C. Lo, and H.C. Wong. A particle swarm optimization-based method for multiobjective design optimizations. *IEEE Transactions on Magnetics*, 41(5):1756–1759, May 2005.

[45] John H. Holland. *Adaptation in Natural and Artificial Systems*. Ann Arbor : University of Michigan Press, 1975.

[46] Jeffrey Horn, Nicholas Nafpliotis, and David E. Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, volume 1, pages 82–87, Piscataway, New Jersey, June 1994. IEEE Service Center.

[47] Xiaohui Hu and Russell Eberhart. Multiobjective Optimization Using Dynamic Neighborhood Particle Swarm Optimization. In *Congress on Evolutionary Computation (CEC'2002)*, volume 2, pages 1677–1681, Piscataway, New Jersey, May 2002. IEEE Service Center.

[48] Xiaohui Hu, Russell C. Eberhart, and Yuhui Shi. Particle Swarm with Extended Memory for Multiobjective Optimization. In *2003 IEEE Swarm Intelligence Symposium Proceedings*, pages 193–197, Indianapolis, Indiana, USA, April 2003. IEEE Service Center.

[49] A. Iorio and X. Li. A cooperative coevolutionary multiobjective algorithm using non-dominated sorting. In K. et al. Deb, editor, *Proceedings of Genetic and Evolutionary Computation Conference*, pages 537–548, Seattle, USA, 2004. LNCS 3102, Springer-Verlag.

[50] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, New Jersey, 1988.

[51] W. Jakob, M. Gorges-Schleuter, and C. Blume. Application of genetic algorithms to task planning and learning. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature, 2nd Workshop*, Lecture Notes in Computer Science, pages 291–300, Amsterdam, 1992. North-Holland Publishing Company.

[52] Stefan Janson and Daniel Merkle. A new multi-objective particle swarm optimization algorithm using clustering applied to automated docking. In María J. Blesa, Christian Blum, Andrea Roli, and Michael Sampels, editors, *Hybrid Metaheuristics, Second International Workshop, HM 2005*, pages 128–142, Barcelona, Spain, August 2005. Springer. Lecture Notes in Computer Science Vol. 3636.

[53] Yaochu Jin and Bernhard Sendhoff. Fitness approximation in evolutionary computation: A survey. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1105–1112. Morgan Kaufmann, 2002.

[54] Nattavut Keerativuttitumrong, Nachol Chaiyaratana, and Vara Varavithya. Multi-objective Co-operative Co-evolutionary Genetic Algorithm. In J.J. Merelo Guervs et al., editor, *Proceedings of PPSN VII*, pages 288–297. Springer-Verlag, September 2002.

[55] James Kennedy and Russell C. Eberhart. Particle Swarm Optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, pages 1942–1948, Piscataway, New Jersey, 1995. IEEE Service Center.

[56] James Kennedy and Russell C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco, California, 2001.

[57] Joshua D. Knowles and David W. Corne. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[58] Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. Combining Convergence and Diversity in Evolutionary Multi-objective Optimization. *Evolutionary Computation*, 10(3):263–282, Fall 2002.

[59] Xiaodong Li. A Non-dominated Sorting Particle Swarm Optimizer for Multiobjective Optimization. In Erick Cantú-Paz et al., editor, *Genetic and Evolutionary Computation—GECCO 2003. Proceedings, Part I*, pages 37–48. Springer. Lecture Notes in Computer Science Vol. 2723, July 2003.

[60] Xiaodong Li. Better Spread and Convergence: Particle Swarm Multiobjective Optimization Using the Maximin Fitness Function. In Kalyanmoy Deb et al., editor, *Genetic and Evolutionary Computation–GECCO 2004. Proceedings of the Genetic and Evolutionary Computation Conference. Part I*, pages 117–128, Seattle, Washington, USA, June 2004. Springer-Verlag, Lecture Notes in Computer Science Vol. 3102.

[61] D. Luce. *Individual Choice Behavior*. Wiley, 1959.

[62] Jiangming Mao, Kotaro Hirasawa, Jinglu Hu, and Junichi Murata. Genetic Symbiosis Algorithm for Multiobjective Optimization Problems. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, page 771, San Francisco, California, July 2001. Morgan Kaufmann Publishers.

[63] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, New York, third edition, 1996.

[64] K. M. Miettinen. *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, Boston, Massachusetts, 1998.

[65] B. L. Miller. *Noise, sampling, and efficient genetic algorithms*. PhD thesis, Department of General Engineering, University of Illinois at Urbana-Champign, Urbana, IL, USA, 1997.

[66] J. N. Morse. Reducing the size of the nondominated set: Pruning by clustering. *Computers and Operations Research*, 7(1-2):55–66, 1980.

[67] Sanaz Mostaghim and Jürgen Teich. The role of ε-dominance in multi objective particle swarm optimization methods. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC'2003)*, volume 3, pages 1764–1771, Canberra, Australia, December 2003. IEEE Press.

[68] Sanaz Mostaghim and Jürgen Teich. Strategies for Finding Good Local Guides in Multi-objective Particle Swarm Optimization (MOPSO). In *2003 IEEE Swarm Intelligence Symposium Proceedings*, pages 26–33, Indianapolis, Indiana, USA, April 2003. IEEE Service Center.

[69] Sanaz Mostaghim and Jürgen Teich. Covering pareto-optimal fronts by subswarms in multi-objective particle swarm optimization. In *2004 Congress on Evolutionary Computation (CEC'2004)*, volume 2, pages 1404–1411, Portland, Oregon, USA, June 2004. IEEE Service Center.

[70] Leandro Nunes de Castro and Jonathan Timmis. *An Introduction to Artificial Immune Systems: A New Computational Intelligence Paradigm*. Springer-Verlag, 2002.

[71] Ender Ozcan and Chilukuri K. Mohan. Analysis of a simple particle swarm optimization system. In *Intelligent Engineering Systems Through Artificial Neural Networks*, pages 253–258, 1998.

[72] Ender Ozcan and Chilukuri K. Mohan. Particle swarm optimization: Surfing the waves. In *Congress on Evolutionary Computation (CEC'1999)*, pages 1939–1944, Washington D.C., USA, 1999. IEEE Press.

[73] J. Paredis. Coevolutionary algorithms. In Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors, *The Handbook of Evolutionary Computation, 1st supplement*, pages 225–238. Institute of Physics Publishing and Oxford University Press, 1998.

[74] Ian C. Parmee and Andrew H. Watson. Preliminary Airframe Design Using Co-Evolutionary Multiobjective Genetic Algorithms. In W. Banzhaf

et al., editor, *Proceedings of GECCO'99*, volume 2, pages 1657–1665, San Francisco, California, July 1999. Morgan Kaufmann.

[75] C. Pimpawat and N. Chaiyaratana. Using a co-operative co-evolutionary genetic algorithm to solve a three-dimensional container loading problem. In *2001 Congress on Evolutionary Computation (CEC'2001)*, volume 2, pages 1197–1204, Seoul, Korea, 2001. IEEE Press.

[76] M. Potter and K. De Jong. A cooperative coevolutionary approach to function optimization. In *Proceedings from the Fifth Parallel Problem Solving from Nature*, pages 530–539, Jerusalem, Israel, 1994. Springer-Verlag.

[77] Tapabrata Ray, Tai Kang, and Seow Kian Chye. An Evolutionary Algorithm for Constrained Optimization. In Darrell Whitley, David Goldberg, Erick Cantú-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2000)*, pages 771–777, San Francisco, California, 2000. Morgan Kaufmann.

[78] Tapabrata Ray and K.M. Liew. A Swarm Metaphor for Multiobjective Design Optimization. *Engineering Optimization*, 34(2):141–153, March 2002.

[79] Margarita Reyes-Sierra and Carlos A. Coello Coello. A study of fitness inheritance and approximation techniques for multi-objective particle swarm optimization. In *Congress on Evolutionary Computation*, pages 65–72, Edinburgh, UK, 2005. IEEE Service Center.

[80] Margarita Reyes-Sierra and Carlos A. Coello Coello. Dynamic fitness inheritance proportion for multi-objective particle swarm optimization. In *Genetic and Evolutionary Computation Conference*, pages 89–90. ACM Press, 2006.

[81] Margarita Reyes-Sierra and Carlos A. Coello Coello. Dynamic fitness inheritance proportion for multi-objective particle swarm optimization. Technical Report EVOCINV-03-2006, Sección de Computación, Departamento de Ingeniería Eléctrica, CINVESTAV-IPN, México, April 2006.

[82] Margarita Reyes-Sierra and Carlos A. Coello Coello. On-line adaptation in multi-objective particle swarm optimization. In *IEEE Swarm Intelligence Symposium*, pages 61–68. IEEE Service Center, 2006.

[83] Margarita Reyes Sierrra and Carlos A. Coello Coello. Un algoritmo coevolutivo para optimización multiobjetivo basado en clustering. In *Segundo Congreso Mexicano de Computación Evolutiva*, pages 63–71, Aguascalientes, Mexico, May 2005.

[84] H. Robbins. Some Aspects of the Sequencial Design of Experiments. *Bulletin of the American Mathematical Society*, 55:527–535, 1952.

[85] Rosin and R. Belew. New methods for competitive coevolution. *Evolutionary Computation*, 5(1):1–29, 1996.

[86] Günter Rudolph. Convergence properties of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, 1(5):96–101, 1994.

[87] Günter Rudolph. On a Multi-Objective Evolutionary Algorithm and Its Convergence to the Pareto Set. In *Proceedings of the 5th IEEE Conference on Evolutionary Computation*, pages 511–516, Piscataway, New Jersey, 1998. IEEE Press.

[88] Mehrdad Salami and Tim Hendtlass. A Fast Evaluation Estrategy for Evolutionary Algorithms. *Applied Soft Computing*, 2(3):156–173, 2003.

[89] Maximino Salazar-Lechuga and Jonathan Rowe. Particle swarm optimization and fitness sharing to solve multi-objective optimization problems. In *Congress on Evolutionary Computation (CEC'2005)*, pages 1204–1211, Edinburgh, Scotland, UK, September 2005. IEEE Press.

[90] Kumara Sastry. Evaluation-relaxation schemes for genetic and evolutionary algorithms. Master's thesis, Department of General Engineering, University of Illinois at Urbana-Champign, Urbana, IL, USA, 2002.

[91] Kumara Sastry, David E. Goldberg, and Martin Pelikan. Don't Evaluate, Inherit. In Lee Spector et.al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 551–558, San Francisco, California, USA, 7-11 2001. Morgan Kaufmann.

[92] J. David Schaffer. *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms*. PhD thesis, Vanderbilt University, 1984.

[93] J. David Schaffer. Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In *Genetic Algorithms and their Applications:*

Proceedings of the First International Conference on Genetic Algorithms, pages 93–100. Lawrence Erlbaum, 1985.

[94] J. R. Schott. Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization. Master's thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1995.

[95] Margarita Reyes Sierra and Carlos A. Coello Coello. A New Multi-Objective Particle Swarm Optimizer with Improved Selection and Diversity Mechanisms. Technical Report EVOCINV-05-2004, Sección de Computación, Departamento de Ingeniería Eléctrica, CINVESTAV-IPN, México, 2004.

[96] Margarita Reyes Sierra and Carlos A. Coello Coello. Coevolutionary Multi-Objective Optimization using Clustering Techniques. In *Fourth Mexican International Conference on Artificial Intelligence (MICAI)*, pages 603–612, Monterrey, Mexico, 2005. Lecture Notes in Computer Science.

[97] Margarita Reyes Sierra and Carlos A. Coello Coello. Fitness Inheritance in Multi-Objective Particle Swarm Optimization. In *IEEE Swarm Intelligence Symposium*, pages 116–123, Pasadena, California, USA, 2005. IEEE Service Center.

[98] Margarita Reyes Sierra and Carlos A. Coello Coello. Improving PSO-based Multi-objective Optimization using Crowding, Mutation and ε-Dominance. In *Third International Conference on Evolutionary Multi-Criterion Optimization, EMO 2005.*, pages 505–519, Guanajuato, México, 2005. LNCS 3410, Springer-Verlag.

[99] Robert E. Smith, B. A. Dike, and S. A. Stegmann. Fitness Inheritance in Genetic Algorithms. In *SAC '95: Proceedings of the 1995 ACM Symposium on Applied Computing*, pages 345–350, Nashville, Tennessee, USA, 1995. ACM Press.

[100] N. Srinivas and Kalyanmoy Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, Fall 1994.

[101] K.C. Tan, Y.H. Chew, T.H. Lee, and Y.J. Yang. A cooperative coevolutionary algorithm for multiobjective optimization. In *IEEE International*

Conference on Systems, Man and Cybernetics, volume 1, pages 390–395. IEEE Press, 2003.

[102] Jürgen Teich, Eckart Zitzler, and Shuvra S. Bhattacharyya. 3D Exploration of software schedules for DSP algorithms. In *7th International Workshop on Hardware/Software Codesign (CODES'99)*, pages 168–172, May 1999.

[103] Gregorio Toscano Pulido and Carlos A. Coello Coello. Using Clustering Techniques to Improve the Performance of a Particle Swarm Optimizer. In Kalyanmoy Deb et al., editor, *Genetic and Evolutionary Computation–GECCO 2004. Proceedings of the Genetic and Evolutionary Computation Conference. Part I*, pages 225–237, Seattle, Washington, USA, June 2004. Springer-Verlag, Lecture Notes in Computer Science Vol. 3102.

[104] Frans van den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa, 2002.

[105] David A. Van Veldhuizen. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*. PhD thesis, Department of Electrical and Computer Engineering. Graduate School of Engineering. Air Force Institute of Technology, Wright-Patterson AFB, Ohio, May 1999.

[106] David A. Van Veldhuizen and Gary B. Lamont. On Measuring Multiobjective Evolutionary Algorithm Performance. In *2000 Congress on Evolutionary Computation*, volume 1, pages 204–211, Piscataway, New Jersey, July 2000. IEEE Service Center.

[107] Joannès Vermorel and Mehryar Mohri. Multi-armed bandit algorithms and empirical evaluation. In J. Gama et al., editor, *Proceedings of the 16th European Conference on Machine Learning*, pages 437–448, Porto, Portugal, 2005. LNAI 3720, Springer-Verlag.

[108] Mario Alberto Villalobos-Arias, Gregorio Toscano Pulido, and Carlos A. Coello Coello. A proposal to use stripes to maintain diversity in a multi-objective particle swarm optimizer. In *Proceedings of the 2005 IEEE Swarm Intelligence Symposium*, pages 22–29, Pasadena, California, USA, June 2005. IEEE Press.

[109] Ivan Voutchkov and A. J. Keane. Multi-objective optimization using surrogates. In *Adaptive Computing in Design and Manufacture. Proceedings*

of the Seventh International Conference, pages 167–175. The Institute for People-Centered Computation (IP-CC), 2006.

[110] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, 1989.

[111] P. B. Wilson and M. D. Macleod. Low implementation cost IIR digital filter design using genetic algorithms. In *IEE/IEEE Workshop on Natural Algorithms in Signal Processing*, pages 4/1–4/8, Chelmsford, U.K., 1993.

[112] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

[113] Zhang Xiao-hua, Meng Hong-yun, and Jiao Li-cheng. Intelligent particle swarm optimization in multiobjective optimization. In *Congress on Evolutionary Computation (CEC'2005)*, pages 714–719, Edinburgh, Scotland, UK, September 2005. IEEE Press.

[114] L.B. Zhang, C.G. Zhou, X.H. Liu, Z.Q. Ma, and Y.C. Liang. Solving Multi Objective Optimization Problems Using Particle Swarm Optimization. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC'2003)*, volume 3, pages 2400–2405, Canberra, Australia, December 2003. IEEE Press.

[115] Eckart Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, November 1999.

[116] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195, Summer 2000.

[117] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland, May 2001.

[118] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In K.C. Giannakoglou et al., editor, *Proceedings of the EUROGEN2001 Conference*, pages 95–100, Barcelona, Spain, 2002. CIMNE.

[119] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, November 1999.

[120] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, April 2003.