CENTER FOR RESEARCH AND ADVANCED STUDIES
OF THE NATIONAL POLYTECHNIC INSTITUTE OF MEXICO
COMPUTER SCIENCE DEPARTMENT

Use of Domain Information to Improve the Performance of an Evolutionary Algorithm

By

Ricardo Landa Becerra

As the fulfilment of the requirement for the degree of

Doctor of Science

Specialization in

Electrical Engineering

Option: Computer Science

Advisor: **Dr. Carlos A. Coello Coello**

Mexico City, Mexico. June, 2007

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL
DEPARTAMENTO DE COMPUTACIÓN

Uso de Información del Dominio para Mejorar el Desempeño de un Algoritmo Evolutivo

Tesis que presenta

Ricardo Landa Becerra

Para obtener el grado de

Doctor en Ciencias

En la especialidad de

Ingeniería Eléctrica

Opción Computación

Director de la Tesis: **Dr. Carlos A. Coello Coello**

México, D.F., junio de 2007

To Norma

To my family

In memoriam
Juan Landa Hernández

Acknowledgments

I wish to thank Dr. Carlos Coello, for his guidance and for sharing his knowledge with me.

I thank my reviewers, whose comments greatly helped me to improve this work: Dr. José Matías Alvarado Mentado, Dr. Luis Gerardo de la Fraga, Dr. Arturo Hernández Aguirre and Dr. Carlos Eduardo Mariano Romero. I also thank Dr. Francisco Rodríguez Henríquez, for his valuable comments which also helped me to improve the contents of this thesis.

I thank my partners at the CINVESTAV, who shared with me some memorable times.

I acknowledge support from CONACyT through a scholarship to pursue graduate studies at the Computer Science Section of the Electrical Engineering Department at CINVESTAV-IPN. I also acknowledge support granted to conclude this thesis through the CONACyT project "Artificial Immune Systems for Multiobjective Optimization" (Ref. 42435-Y), whose principal investigator is Dr. Carlos A. Coello Coello.

Abstract

In this thesis we explore the use of domain information incorporated during the execution of an evolutionary algorithm, through the use of a cultural algorithm. The cultural algorithms are evolutionary algorithms that support an additional mechanism for information extraction during the execution of the algorithm, avoiding the need to encode the information a priori.

Firstly, a cultural algorithm to tackle constrained optimization problems was developed. Such algorithm adopts differential evolution as its model for the population. Using the differential evolution operators as a base, we designed four knowledge sources, each one with a particular influence over the operators. Since each knowledge source exhibits different benefits in different phases of the search, a main mechanism to control the application rate of the operators was developed, based on the success rate each one.

This algorithm was tested using a well-known benchmark and a pair of instances of engineering optimization problems, and compared with other representative algorithms of the state-of-the-art. In both cases, equal or better solutions were obtained, requiring a smaller number of objective function evaluations.

In the next phase, a hybrid algorithm to tackle multiobjective optimization problems was developed. Such algorithm is a hybrid between the previous algorithm for constrained optimization, and a method of mathematical programming called ε-constraint. We obtained other advantages with this algorithm, like obtaining good approaches of the Pareto front in problems that have been very difficult to solve for other evolutionary approaches.

As a last contribution, we introduced an approach to perform incor-

poration of preferences to the previous algorithm, but such approach can also be used in an wide set of techniques. This proposal is based on the use of vectors of goals. With the addition this approach, is possible to reduce the computational cost needed when applying the hybrid algorithm on problems with a large number of objectives, turning it applicable on practice.

Resumen

En esta tesis se explora el uso de información del dominio incorporada durante la ejecución de un algoritmo evolutivo mediante un algoritmo cultural. Los algoritmos culturales son algoritmos evolutivos que soportan un mecanismo adicional para la extracción de información durante la misma ejecución del algoritmo, eliminando de esta forma la necesidad de codificar la informacion *a priori*.

Primeramente, se desarrolló un algoritmo cultural para atacar problemas de optimización con restricciones. Tal algoritmo utiliza evolución diferencial como modelo para la población. Utilizando como base los operadores de la evolución diferencial, se diseñaron cuatro fuentes de conocimiento, cada una con una influencia particular sobre los operadores. Debido a que cada fuente de conocimiento presenta diferentes beneficios en diferentes fases de la búsqueda, se implementó un mecanismo maestro para controlar la frecuencia de aplicación de los operadores, basandose en la tasa de éxito de cada uno.

Este algoritmo fue probado con un conjunto de problemas de prueba bien conocido y con un par de instancias de problemas de optimización de ingeniería, y comparados con otros algoritmos representativos del estado del arte. En ambos casos, se obtuvieron iguales o mejores soluciones, requiriendo un número menor de evaluaciones de la función objetivo.

En la siguiente fase, se desarrolló un algoritmo híbrido para atacar problemas de optimización multiobjetivo. Tal algoritmo es un híbrido entre el algoritmo previo para optimización con restricciones, y un metodo de programación matemática llamado restricción-ε. Con este algoritmo se obtuvieron otras ventajas, como obtener buenas aproximaciones del frente de Pareto en problemas que se han mostrado muy dificiles de resolver por otras técnicas evolutivas.

Como un último aporte se presentó una propuesta para realizar incorporación de preferencias al algoritmo previo, pero que se puede utilizar en un conjunto amplio de técnicas. Esta propuesta está basada en el uso de vectores de metas. Con la incorporación de esta propuesta, es posible reducir el costo computacional necesario al emplear el algoritmo híbrido en problemas con un número elevado de funciones objetivo, volviéndolo aplicable en la práctica.

Contents

List of Figures

List of Tables

List of Algorithms

1

Introduction

Evolutionary Algorithms are a very important class of optimization metaheuristics, popular because of its innovative concepts, and its good results in many problems. Another important characteristic of evolutionary algorithms is the flexibility to be applied in a wide range of problems, with little or no changes at all within the algorithm (except for the objective function). However, as the range of applications increases, it increases also the need for improving the convergence rates of the algorithm, even sacrificing the flexibility of the algorithm.

Some authors have proposed the use of domain information to improve the performance of the algorithms, obtaining very encouraging results. Some other researchers have proposed that the use of domain information is one of the alternatives to circumvent the limitations of the no free lunch theorem [176]. The strategies to add domain information to an evolutionary algorithm can be diverse. There exist proposals to add information during the evaluation, during the application of evolutionay operators, or during the selection mechanism, among others.

A particular strategy to add domain information to an evolutionary algorithm is the use of cultural algorithms. This class of algorithms are inspired in the cultural evolution of some social species, paricularly humans, which use their ability to and share their knowledge to perform a faster adaptation to their environment. In cultural algorithms, the information is acquired by the individuals (solution trials) generated during the search process, and such information is share by the entire popula-

tion (group of current solutions), in order to modify its characteristics for a better adaptation (fitness, or solution quality).

In order to explore the capabilities of cultural algorithms, two particular classes of problems are tacked in this thesis work. They are the constrained single-objective optimization, and multiobjective optimization.

Constrained optimization is a class within global optimization problems, whose feasible region is restricted for a group of constraints. Solutions that are infeasible, cannot be considered valid. These problems are very common in some disciplines, as well as in practical situations which involve optimization tasks. Because of that reason, these problems have been tacked with many methods, including mathematical programming as well as metaheuristics.

Multiobjective optimization is a generalization of global optimization problems, where two or more objective functions are allowed. The objective functions are often in conflict, and thus a single optimum for all the objective functions is not possible.

In both cases, the current research has been obtained very good results, and now one of the main concerns is to improve the convergence capabilities of the algorithms, reducing the number of objective function evaluations needed to obtain good results, or alternatively to provide better approximations of the global optima for the most challenging problems existing in the standard benchmarks. As stated previously, cultural algorithms are a promising alternative when such goal is pursued.

In this thesis work, we try to validate the hypotesis of performance improvement of cultural algorithms, for the specific cases of constrained optimization and multiobjective optimization.

For the case of constrained optimization, a cultural algorithm will be developed with the aim of increasing the convergence rates (more precisely, of decreasing the total number of objective function evaluations needed to obtain an approximation of the global optimum) when compared with other state-of-the-art algorithms, obtaining similar quality results.

For the case of multiobjective optimization, a hybrid algorithm of the previous proposal with a mathematical programming method will be developed. In this case, we pretend to exploit the characteristics of a fast approximation to the global optimum, together with a method which requires several constrained single-objective optimizations. If the constrained single-objective approach performs a good approximation, the

entire method will be able to well approximate the solution to the multiobjective problem, even in cases when other approaches that produce multiple points at a time (as most evolutionary approaches) present difficulties.

The rest of the document is organized as follows:

In Chapter 2 are presented the basic concepts about optimization and evolutionary computation, including the main paradigms within this area; and the techniques in which this work is based.

Chapter 3 includes a brief review of the state-of-the-art in evolutionary approaches developed to solve constrained optimization problems. A special emphasis is made for the approaches based on cultural algorithms, and on differential evolution, because the relevance they have in this work.

In Chapter 4, the new approach for constrained optimization is presented, detailing its main mechanisms.

Chapter 5 contains the results obtained by our approach presented in the previous chapter, and a comparison with other approaches.

Chapter 6 includes a brief review of the state-of-the-art in evolutionary approaches developed to solve multiobjective optimization problems.

In Chapter 7 is presented the hybridization of a mathematical programming method with our approach for constrained optimization, in order to solve multiobjective problems. An additional mechanism is also introduced with the aim of reducing the overall objective function evaluations.

In Chapter 8 are contained the results of our approach presented in the previous chapter, for experiments performed with and without the additional mechanism. In both cases, the results are compared with other approach.

In Chapter 9 is introduced a methodology to incorporate preferences of the decision maker into our proposed technique. The results of some experiments are also included.

Finally, Chapter 10 includes our conclusion and some ideas of future work in the topics related to this thesis.

2

Background

2.1 Optimization

Optimization is an area from mathematics that has many applications in real life. Optimization problems appear constantly in industrial processes, engineering, finance, and other human activities.

An optimization problem consists of the following elements:

- *Decision variables*. These are the values that we modify in order to solve the problem, and are denoted by the vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. In this thesis, we will only deal with problems in which the decision variables are real numbers, $\mathbf{x} \in \mathbb{R}^n$.

- *Objective functions*. These are m functions of the independent variable $\mathbf{x}$, $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x}))$, with $m \geq 1$. The result of their evaluations is what we want to optimize (find a minimum or maximum). In this work we only deal with real valued functions, $f_i : \mathbb{R}^n \mapsto \mathbb{R}$.
  If $m = 1$, the problem is called a *single objective optimization problem*.
  If $m \geq 2$, the problem is a *multiobjective optimization problem*.

- *Constraints*. Equalities and/or inequalities containing some or all of the decision variables, that must be satisfied by a solution, so that

such solution is considered feasible, and therefore, a valid solution of the problem.[1]

A solution is called *infeasible* if it violates at least one constraint. Otherwise, it is called *feasible*.

Given its components, the problem is defined as follows: find the vector(s) $\mathbf{x}$ such that

$$\begin{array}{ll} \text{minimize} & \mathbf{f}(\mathbf{x}) \\ \text{subject to} & \mathbf{x} \in S \end{array}$$

where $S \subseteq \mathbb{R}^n$ is the feasible region defined by the constraints:

$$\begin{array}{l} g_i(\mathbf{x}) \leq 0,\ i = 1, \ldots, \ell \\ h_j(\mathbf{x}) = 0,\ j = 1, \ldots, k \end{array}$$

where g_i are the ℓ inequality constraints, and h_j are the k equality constraints.

In the previous definition, only minimization problems are considered. If a problem includes an objective function f_i to be maximized, then we can transform the problem into the minimization of $-f_i$, without loss of generality.

Another common practice is to transform the equality constraints of the form

$$h_j(\mathbf{x}) = 0$$

into two inequality constraints:

$$\begin{array}{rcl} g_{2j-1}(\mathbf{x}) & \leq & 0 \\ g_{2j}(\mathbf{x}) & \geq & 0 \end{array}$$

or into a relaxed form, which facilitates finding feasible solutions:

$$\begin{array}{rcl} g_{2j-1}(\mathbf{x}) & \leq & \epsilon \\ g_{2j}(\mathbf{x}) & \geq & -\epsilon \end{array}$$

using a small value for ϵ (0.01 or less).

[1] This definition includes only *hard constraints*. There also exist the so called *soft constraints*, that are desirable to fulfill, but not mandatory. If a solution violates one or more soft constraints, is still considered valid as long as it does not violate any of the hard constraints. In this thesis, we only deal with hard constraints.

The first optimization problems that were widely studied and solved, were those of linear programming. Linear programming includes all the problems in which all the objective functions and constraint functions are linear. Currently, there exist several methods to solve linear programming problems, in an efficient and effective manner. For example, the simplex algorithm [34].

If an objective function and/or a constraint function is nonlinear, then the problem is nonlinear. For that case, there are also mathematical programming methods, but most of them require additional information about the problem: for example, the first derivative is required for many methods. This information is not always available for the application we are dealing with (the objective function may not be differentiable).

For the general case, a method to find the global optimum in polynomial time is unavailable. In order to provide a solution for these types of problems in practical times, the alternative is to develop metaheuristics, and try to obtain the best possible performance [143].

With the help of modern digital computers, recent research in nonlinear and combinatorial optimization has been following that direction. New algorithms are developed to perform a high number of operations, and a high number of iterations are frequently executed, due to the availability of faster modern computers. This is done with the aim of improving the quality of the solutions found at previous iterations.

Currently, there exist several paradigms developed in the area of metaheuristics, and each of them with several variants, to solve a variety of problems. It is worth mentioning simulated annealing [85], and tabu search [56], as important examples of metaheuristics.

A group of metaheuristics which is currently very popular, is evolutionary computation (mainly genetic algorithms), mainly due to the results obtained by evolutionary computation methods in a variety problems, which are sometimes better than those of any other technique previously adopted to solve such problem.

2.2 Evolutionary computation

Evolutionary computation is based on the ideas of natural selection, and its application on an artificial environment [50, 57]. Natural selection is seen as an optimization process [10], in which the individuals of a popu-

lation gradually adapt to their environment.

For evolutionary computation algorithms, an individual is a potential solution to a problem, encoded according to the structure of the algorithm; and the environment to which the individual belongs is the objective function(s) and the constraints; such an environment will determine the survival capability of an individual.

As in natural environments, evolutionary algorithms work with a population, i.e., with several solutions at a time, contrary to other metaheuristics which work with a single solution. This feature gives evolutionary algorithms the ability to escape from local optima more easily.

During the execution of an evolutionary algorithm, some (mainly probabilistic) variation operators are applied to the population, in order to obtain new solutions, which are preserved or discarded through a selection mechanism. This process is repeated by a certain number of iterations, that we call generations. This number of generations can be defined by the user or obtained by the algorithm itself.

The most common variation operators incorporated into an evolutionary algorithm are crossover or recombination, and mutation. Crossover consists of the combination of two or more individuals (parents), to obtain one or more new individuals (children). Mutation consists of a (normally small) alteration of an individual.

Such are the shared features among the three main paradigms of evolutionary computation. A further description of these paradigms is provided in the following sections.

2.2.1 *Evolutionary programming*

It was initially proposed by Lawrence J. Fogel [51], with the aim to produce finite state machines. Later on, David Fogel proposed a version of evolutionary programming for numeric optimization [49].

The evolutionary programming algorithm is shown in Algorithm 1. The Initial population P is of size μ. The mutation operator obtains one child from each individual in the population (μ children are obtained at each generation, and they conform P'), and it is based on a random variable with normal distribution. The standard deviation of the random variable can be defined *a priori*, but there are procedures to self-adapt its value for each variable.

Algorithm 1: Evolutionary programming

```
initialize(P = {x_1, ..., x_μ})
evaluate(P)
repeat
  for j = 1 to μ do
    for i = 1 to n do
      x'_{i,j} = N(x_{i,j}, σ)
    end for
  end for
  evaluate(P')
  P = selection(P ∪ P')
until the termination condition is achieved
```

Crossover is not applied in this case, because evolutionary programming simulates the evolutionary process at the species level, and different species cannot recombine between them.

Selection is performed through stochastic tournaments, taking into account the μ parents and the μ children. The operator selects again μ individuals, which will conform the population for the next generation.

2.2.2 *Evolution strategies*

Developed by Ingo Rechenberg [142], who first proposed the $(1+1)$ evolution strategy (or $(1+1)$-ES), which works with a single individual. Later on, appeared the $(\mu/\rho, \lambda)$-ES and the $(\mu/\rho + \lambda)$-ES, which adopt a population. Usually, if $\rho = 2$, the letter ρ is omitted from the name.

The algorithm of evolution strategies is shown in Algorithm 2. The initial population is of size μ, and the recombination operator randomly selects a set Q of ρ parents to generate each child. A total of λ children are generated. Then, each child is mutated through a normal distributed random variable, whose standard deviation can be self-adapted.

If the strategy is a $(\mu/\rho, \lambda)$-ES, then the selection operator will only consider the λ children to obtain the new generation, while if the strategy is a $(\mu/\rho + \lambda)$-ES, then the selection operator will obtain the new generation from both, parents and children.

The selection operator is deterministic (different from the evolutionary

Algorithm 2: Evolution strategies

```
initialize(P = {x_1, ..., x_μ})
evaluate(P)
repeat
  for j = 0 to λ do
    Q = sel_parents(ρ, P)
    x'_j = crossover(Q)
  end for
  for j = 1 to λ do
    for i = 1 to n do
      x'_{i,j} = N(x'_{i,j}, σ)
    end for
  end for
  evaluate(P')
  if Strategy type is comma then
    P = selection(P')
  else if Strategy type is plus then
    P = selection(P ∪ P')
  end if
until the termination condition is achieved
```

programming operator), and that means that the best μ individuals survive for the next generation. For that reason, the $(\mu/\rho+\lambda)$-ES is frequently called *elitist* strategy, which means that it never loses the best individual found.

The mutation operator is based on a normal distributed random variable, as the evolutionary programming operator. There exist several recombination operators for the evolution strategies, but this operator is a secondary one, and is sometimes omitted.

2.2.3 *Genetic algorithms*

They were proposed by John Holland [66] with the aim to solve machine learning problems. Genetic algorithms are the most popular paradigm of evolutionary algorithms, perhaps because of their flexibility to solve a wide variety of problems.

Algorithm 3: Genetic algorithm

```
initialize(P = {x_1, ..., x_μ})
evaluate(P)
repeat
  Q = sel_parents(P)
  P' = crossover(Q)
  for j = 1 to μ do
    x'_j = mutation(x'_j)
  end for
  evaluate(P')
  P = P'
until the termination condition is achieved
```

The genetic algorithm is shown in Algorithm 3. In genetic algorithms is common to encode solutions using binary strings, in opposition to evolution strategies and evolutionary programming, which normally use no encoding.

However, there exist several proposals for representing solutions in genetic algorithms, as well as a variety of selection, crossover and mutation operators [2]. Because of this variety of options, we do not detail any of these operators in Algorithm 3.

A singular feature in genetic algorithms is that the crossover and mutation operators are only applied a certain number of times (these are user-defined parameters). This way, the probability of crossover (generally between 0.7-1.0) is used to determine if recombination is applied or not; and the probability of mutation (commonly 0.1 or less) is used for the application of the second operator. Considering the typical values used in normal practice, is evident that crossover is the main operator of genetic algorithms.

2.3 Knowledge incorporation in evolutionary computation

Evolutionary algorithms were initially designed, and vastly applied, to problems with very large (and possibly accidented) search spaces. Such search spaces are normally too large (and therefore, untractable) for classical optimization methods, due to fact that there is no further knowledge about the problem available, or it is too difficult to incorporate such knowledge into the given method.

When designing a metaheuristic, the developer must choose a compromise between flexibility and speed. The techniques that favor speed are, for example, knowledge-based systems, while on the other hand, the extreme of flexibility is the exhaustive search.

Most evolutionary computation techniques emphasize flexibility, solving a wide variety of problems, but adding only a small amount of domain knowledge: commonly the only knowledge exploited by an evolutionary algorithm is the one contained in the fitness function [140].

One of the early attempts to add domain knowledge within evolutionary algorithms is the approach called EnGENEous, proposed by Powell et al. [135], in which an expert system works together with a genetic algorithm. Unfortunately, this approach was found to be inefficient for some applications.

The same authors continued working with the integration of expert systems and genetic algorithms, for example in the approach called *Interdigitation* [137], in which they also integrate some methods of numerical optimization in order to better approximate the global optimum of a problem.

Louis and Rawlins proposed to incorporate domain knowledge into evolutionary algorithms for solving several design problems (logic circuits design and design of trusses, among others). First, they added domain knowledge only in the recombination operators [108, 109]. Later on, they also incorporated knowledge to the initialization of the population, to the encoding of solutions, and to the other evolutionary operators [110].

Another example of problem knowledge incorporation are the cultural algorithms, proposed by Robert Reynolds [146]. These algorithms are a particular type of evolutionary algorithms, in which the domain knowledge is not encoded *a priori* to the technique, but is extracted during the search process itself [146].

2.4 Cultural algorithms

Reynolds developed the cultural algorithms as a complement to the metaphor which inspired the evolutionary algorithms (natural selection and genetic concepts) [146].

The cultural algorithms are based on some sociological and anthropological theories, which have tried to model the phenomenon called *cultural evolution*. Such theories propose that the evolution of societies where culture exists, is slightly more complicated than only genetic evolution, and it can be seen as a process of inheritance at two levels: the micro-evolutionary level, which consist on the genetic material inherited by parents to their descendants; and the macro-evolutionary level, which is conformed by the knowledge acquired by the individuals through their experiences, that once encoded and stored, is useful for guiding the behavior of new individuals in a population (not only descendants in a genetic line) [145, 44].

Culture, then, can be seen as a set of ideological phenomena shared by a population, that influences the way an individual interprets its experiences and decides its behavior. *Culture affects the success and survival of individuals and groups, leading to evolutionary processes that are every bit as real and important as those that shape genetic variation* [151].

In these models is easy to appreciate the component of the system that is shared by the population: the knowledge, collected by the members of a society, but encoded in a way that is potentially accessible for all the population. Similarly, the individual components of the system are the

experiences, and the way they can contribute to the shared knowledge, for the other individuals to learn them indirectly.

Reynolds adopted this phenomenon of double inheritance, as inspiration to create cultural algorithms [146]. The aim is to increase the convergence or learning rates, and therefore, that the system responds better to a variety of problems [54].

The components of cultural algorithms are the following:

The population space. As other evolutionary algorithms, this space maintains a set of individuals (potential solutions to the problem). Each individual possesses characteristics independently from other individuals, and these characteristics define its fitness in the environment (the problem to solve). Through the generations, individuals can be replaced by their descendants, obtained by means of the application of operators that somehow affect the population.

The belief space. In this space is where the knowledge, acquired by the individuals through the generations, will be stored. This knowledge must be accessible to any individual in the population, and can be used to influence its behavior (modify its characteristics and then modify its fitness).

A communication protocol is necessary to link both spaces, defining rules about the type of information that the spaces will interchange (*i.e.*, which information will pass from the population space to the belief space and vice versa).

The main steps of a cultural algorithm are shown in Algorithm 4.

Most of the steps of a cultural algorithm are similar to those of a standard evolutionary algorithm, and, as it can be seen, the differences are found on the steps that include the belief space B. We will perform a review of those steps to state the differences.

The third step is the initialization of the belief space. The details of this procedure depend on the structure of the belief space, and may be different in each case. In the next chapters is described the initialization of the belief space for the particular case of the approach proposed here.

In the main loop is the `update` (also called `adjust`) function of the belief space. In this step is when the algorithm incorporates new experiences to the belief space, from a selected group of individuals. The function

Algorithm 4: Cultural algorithm

```
initialize(P = {x_1, ..., x_μ})
evaluate(P)
initialize(B)
repeat
  update(B, accept(P))
  P' = influence(operators(P))
  evaluate(P')
  P = selection(P')
until the termination condition is achieved
```

`accept` selects the individuals of this group, which are chosen from the population.

On the other hand, the variation operators of the algorithm (such as the recombination and mutation) are modified by the function `influence`. This function introduces a bias for the children resulting from the variation operators so that they get closer to the desirable behaviors, and farther away from the undesirable ones, according to the information stored in the belief space.

The functions `accept` and `influence` are the components of the communication protocol, because the information flows through them linking the population and belief spaces. These interactions are graphically shown in Figure 2.1 [147].

In [146], Reynolds adopted the population of a genetic algorithm (together with its operators) as the population space of his proposal, and the version spaces [119] as the belief space. This cultural algorithm was called *Version Space guided Genetic Algorithm*, (*VGA*), and was applied to solve some instances of the Boole problem[2] [175], with encouraging results.

In the same work, Reynolds tries to show the usefulness of cultural algorithms with an adaptation of the schema theorem, taking advantage of the genetic algorithm-based approach previously introduced. The schema theorem is an expression that bounds the propagation of the best schemes among the population of a genetic algorithm [67].

This modification indicates that a genetic algorithm, with the addition

[2]This problem consists of infering the characteristic function for an unknown boolean multipliexer.

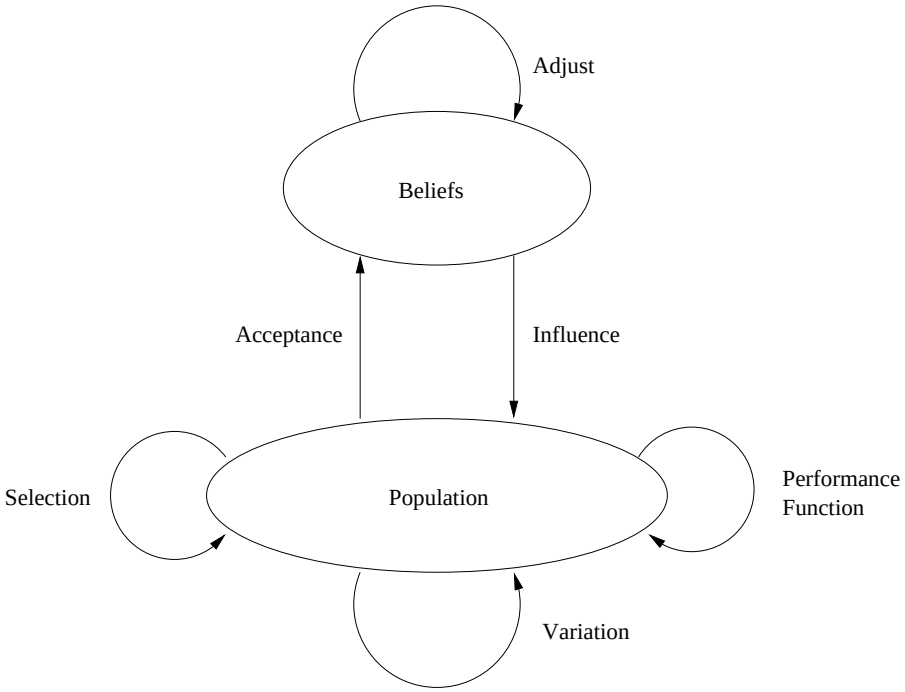


Figure 2.1: Spaces of a cultural algorithm

of a belief space, can improve its performance, increasing its convergence rates.

In this thesis we investigate this claim of performance improvement, in another evolutionary algorithm. Also, we designed the new cultural approach to be applied in problems scarcely tackled with cultural techniques, such as the constrained optimization problems. Finally, we hybridize a cultural algorithm with a mathematical programming technique, to solve the multiobjective optimization problems.

2.5 Differential evolution

Differential evolution is a recently developed evolutionary algorithm originally proposed by Price and Storn [138], whose main design emphasis is real parameter optimization. Differential evolution is based on a mutation operator, which adds an amount obtained by the difference of two individuals randomly chosen from the current population, in contrast with most evolutionary algorithms, in which mutation is performed through a random variable.

Algorithm 5: Differential evolution (DE/rand/1/bin version)

`initialize`$(P = \{\mathbf{x}_1, \ldots, \mathbf{x}_\mu\})$
`evaluate`(P)
repeat
  for $j = 0$ to μ **do**
    Let r_1, r_2 and r_3 be three random integers in $[1, \mu]$, with $r_1 \neq r_2 \neq r_3$
    Let i_{rand} be a random integer in $(1, n)$
    for $i = 1$ to n **do**

$$x'_{i,j} = \begin{cases} x_{i,r_3} + F * (x_{i,r1} - x_{i,r2}) & \text{if } U(0,1) < CR \text{ or } i = i_{rand} \\ x_{i,j} & \text{otherwise} \end{cases}$$

    end for
    `evaluate`$(\mathbf{x}'_j)$
    if `isbetter`$(\mathbf{x}'_j, \mathbf{x}_j)$ **then**
      $\mathbf{x}_j = \mathbf{x}'_j$
    end if
  end for
until the termination condition is achieved

The basic algorithm of differential evolution is shown in Algorithm 5; F and CR are parameters given by the user, and $U(a, b)$ is a realization of a uniformly distributed random variable between a and b. The function $\texttt{isbetter}(\mathbf{x}_a, \mathbf{x}_b)$ returns *true* if $\mathbf{x}_a$ is better than $\mathbf{x}_b$, considering a given criterion (commonly, the value of the objective function).

The algorithm's main component is the variation operator, which consists of a weighted difference of the decision variables of two random individuals, added to the same variable of a third randomly chosen individual. This operator is applied with a probability of CR (similarly to the probability of crossover of a genetic algorithm, user defined), but it is applied at least once for each individual (this is controlled by the parameter $i_{\texttt{rand}}$ in Algorithm 5).

Another important part of differential evolution is the selection operator, which performs the selection progressively during the generation of children, making only local comparisons. The function `isbetter` is adopted to test the condition for replacement of the parent $\mathbf{x}_j$ for the child $\mathbf{x}'_j$. These local comparisons make the algorithm more efficient, avoiding the need of sorting or ranking the population.

The authors of the differential evolution algorithm have suggested that by computing the difference between two individuals randomly chosen from the population, the algorithm is actually estimating the gradient in that zone (rather than in a point). This approach also constitutes a rather efficient way to self-adapt the mutation operator (as a matter of fact, the differential evolution algorithm is also capable of self-adapting both the step sizes and the step direction, since they both depend of the current solutions in the population). Furthermore, when adopting $CR = 1$, the variation operator is rotationally invariant [138], which means that the differential evolution algorithm supports self-adaptation at the same level that an evolution strategy with covariance matrices to self-adapt its standard deviations (this is the most complete self-adaptation scheme available in evolution strategies [159]).

The version of differential evolution shown in Algorithm 5, is called DE/rand/1/bin, and is recommended to be the first choice when trying to apply differential evolution to any given problem [138]. That is the reason why we adopted it for the work reported here. However, there are some other versions of the differential evolution algorithm, and the modifications made here to the variation operator may have certain similarities with some of those versions, as we will note later on.

3

Constraint-Handling Techniques used with Evolutionary Algorithms

In this chapter, we will first briefly review the constraint handling techniques adopted in evolutionary computation in general. Then, we will review the constrained optimization proposals for cultural algorithms and for differential evolution, which are the techniques on which this work is based.

3.1 Constraint-handling techniques

3.1.1 *Penalty functions*

Historically, the most common way to deal with constrained optimization problems in evolutionary computation has been, the penalty functions. Such approach consist on modifying the objective function, adding or substracting it a penalty term, whose magnitude depends on the amount of constraints violation. This way, the constrained problem is transformed into an unconstrained one.

Penalty approaches can be classified into two categories: interior penalties and exterior penalties [139].

- *Interior penalties.* They are designed to perform the search only inside the feasible region. The function penalizes in a small amount the

solution that are far from the frontier of the feasible region, and the penalty increases as the solution gets nearer to the frontier.

- *Exterior penalties.* The search can start from the infeasible region, and the penalty decreases as the solution approaches the feasible region.

Interior penalties are not very common in evolutionary computation, mainly because they have an important disadvantage: they require a starting feasible solution. For that reason, we only analyze exterior penalties.

The modified objective function for its use with exterior penalties approaches is the following:

$$\phi(\mathbf{x}) = f(\mathbf{x}) \pm \left(\sum_{i=1}^{n} r_i G_i(g_i(\mathbf{x})) + \sum_{j=1}^{p} c_j H_j(h_j(\mathbf{x})) \right)$$

where ϕ is the new objective function, G_i is a function for each inequality constraint g_i, H_j is a function for each equality constraint h_j, and r_i and c_j are the penalty factors for each constraint.

In [21], the author enumerates the most common functions adopted for G_i and H_j:

$$\begin{aligned} G_i(g_i(\mathbf{x})) &= \max(0, g_i(\mathbf{x}))^{\beta} \\ H_j(h_j(\mathbf{x})) &= |h_j(\mathbf{x})|^{\gamma} \end{aligned}$$

where the exponents β and γ are commonly 1 or 2 ($\beta, \gamma \in \{1, 2\}$).

Richardson et al. [149] studied the way the penalty functions must be designed, searching for the most important issues to take into account. From their work, we found the following conclusions:

1. The penalty functions that depend on the distance to the feasible region, have a better performance than those that only depend on the number of violated constraints.

2. In problems with a small number of constraints and a small feasible region, penalty functions that depend only on the number of violated constraints, probably will not find feasible points.

3. Good penalty functions must consider the maximum completion cost and the expected completion cost (completion cost is related to the distance to the feasible region).

4. Good penalty functions must be close to the expected completion cost, but not fall often below it. Otherwise, it is possible that no feasible solutions will be found.

It is possible to classify the penalty functions according to the information that they use to adapt their value in time. The first type are the *static* penalty functions, which maintain their level of penalty constant during all the execution. These functions often require further parameters (as the penalty factors) that are dependent of the problem we are dealing with.

On the other hand, the penalty function can take the number of current generation as an argument, to adapt its value; if that is the case, then the penalty function is *dynamic*. This type of functions regularly start the evolutionary process with a low penalty, to diversify the search; but they increase the penalty level during the search, to conclude with a high penalty, in order to obtain a feasible solution. Kazarlis and Petridis [82] performed a series of experiments on dynamic penalty functions, and they concluded that the quadratic functions perform better than others.

A special case of dynamic functions are those based on simulated annealing (*e.g.*, [113]), which increment their level of penalty through the evolutionary process according to a cooling schedule.

Dynamic penalty functions have the same problems of the static penalty functions, since both require parameters that are problem dependent.

The last type of penalty functions are the *adaptive* functions, in which the level of penalty adapts according to the situations of the current population. Some examples of adaptive penalty function approaches are [4, 59], where the penalty factor is increased if the best individual of the recent generations was always feasible, it is decreased if such individual was always infeasible, and remains the same if the best individual was sometimes feasible and sometimes infeasible.

It is worth mentioning three more penalty approaches, that don't fit well into any of the above classifications. The first is the coevolutionary penalty of Coello Coello [20], where two populations are maintained: one with individuals encoding the decision variables of the problem; and the other that encodes the penalty factors. This way, the solutions of the problem and the penalty factors are evolved at the same time. In this algorithm, the penalty function depends on both the number of violated constraints, and on the amount of constraints violation.

The second approach is the segregated genetic algorithm of Le Riche

et al. [150]. In this algorithm, the authors propose the use of two penalty factors per each constraint, one high and one low, in two populations that interact with each other. The aim of the use of two penalty factors is to provide both strong and weak penalties.

Finally, the adaptive segregational constraint-handling evolutionary algorithm (ASCHEA) by Hamida and Schoenauer [64], is based on three components: (1) an adaptive penalty function, (2) a constraint-driven recombination, and (3) a segregational selection based on feasibility. In ASCHEA's most recent version [64], the authors propose to use a penalty factor for each constraint of the problem. Also, the authors added a niching mechanism [40] to improve the performance of the algorithm in multimodal functions. Finally, the authors added a dynamic and an adaptive scheme to decrease the tolerance value used in the transformation of equality constraints into two inequality constraints. The approach is based on an evolution strategy with standard arithmetical recombination.

The rest of the review of constraint handling techniques is divided according to the classification from [21].

3.1.2 *Special representations and operators*

Some problems may appear particularly difficult for evolutionary algorithms with normal representations. In such cases, it may be useful to design a special representation for the problem at hand, and, if necessary, a set of special operators to work with such representation.

Davis [36], for example, introduces several special representations and operators to deal with hard problems of the real world (robot path generation, scheduling optimization, synthesis of neural networks and analysis of DNA).

The random keys, a special representation designed to solve combinatorial problems such as the job assignment on parallel machines, were proposed by Bean [3]. With the use of random keys, it is possible to solve combinatorial problems with real numbers encoding, because they always produce feasible solutions, avoiding the need of special operators.

An important example of special operators for this work, is the first version of GENOCOP [114]. This algorithm was designed to solve constrained problems, but only with linear constraints. It starts with at least one feasible solution, and the operators produce new points that are linear combinations of the parents. Later on, the authors proposed new versions

of GENOCOP, to deal with nonlinear constraints, but they do not fit into this classification of special operators.

Another proposal with special operators is the constraint consistent genetic algorithm of Kowalczyk [88]. The operators, as well as a mechanism to initialize the population, are used to keep the solutions consistent with respect to the constraints. However, such mechanisms are very expensive, computationally speaking.

Schoenauer and Michalewicz [157] developed some operators to explore the bounds between the feasible and the infeasible regions. The oscillation between regions is convenient, because in many problems the global optimum is found in the boundary of the feasible region. However, these operators are highly specialized, and must be designed for a particular problem.

A group of techniques based on special representations are the decoders. The decoders consist of a transformation from a feasible solution to an encoded solution, and the transformation must be designed in such a way, that any possible solution in the algorithm can be mapped to a feasible solution. The most relevant decoders-based technique, is the homomorphous mappings (HM) of Koziel and Michalewicz [89, 90]. In this proposal, the feasible region is mapped into an n-dimensional hypercube, and is able to work with any type of constraints, convex and nonconvex feasible regions, and even disjoint regions. The main idea of this approach is to transform the original problem into another (topologically equivalent) function that is easier to optimize by an evolutionary algorithm. This approach uses a binary-coded GA with Gray codes, proportional selection without elitism and traditional crossover and mutation operators. In the original source, this technique was tested with a test suite proposed by Michalewicz and Schoenauer [116], which later became the standard benchmark for evolutionary constrained optimization; in this benchmark, the homomorphous mappings produced better results than any other evolutionary algorithm, and because of that, the algorithm became a reference for the constraint handling mechanisms designed for evolutionary algorithms within the next few years after its publication.

3.1.3 *Repair algorithms*

This type of algorithm has been widely used for combinatorial optimization problems, where invalid permutations are repaired in order to be-

come valid. Moreover, they have shown to be very good performers in this type of problems [103, 104].

A mechanism of solutions repairing can be found in a more recent version of GENOCOP (GENOCOP III [115]). This version works with two sub-populations: the first one contains the points of the search, and, as in the original GENOCOP, feasiblity is maintained with respect to the linear constraints of the problem; the other sub-population keeps some reference feasible points. Before the evaluation, the points of the first population are repaired using the points on the second population, in order to make them feasible with respect to the nonlinear constraints.

3.1.4 *Separation of objectives and constraints*

Some authors propose to make an explicit separation between objectives and constraints. For instance, Paredis [129] proposed another coevolutionary algorithm with two sub-populations: the first sub-population maintains the generated solutions, and the second one contains one individual for each constraint. The individuals of the second population are evaluated according to the number of individuals on the first population that violate the constraints it represents.

Superiority of feasible points is another form of separation of objectives and constraints, and this concept has been applied by several authors. For example, Powell and Skolnick [136] proposed a method where the feasible points always have a fitness between $-\infty$ and 1, and the infeasible points always have a fitness between 1 and $+\infty$.

A similar technique was proposed by Deb [38], where the infeasible points always have a worst fitness than the feasible ones. This is done by adding the amount of violation of an individual to the fitness of the worst feasible individual in the population. Such fitness assignment is made simpler with the use of binary tournaments, according to the following rules:

1. If an individual is feasible, and the other is infeasible, the feasible one always wins.

2. If both individuals are feasible, the winner is the one with a better value of the objective function.

3. If both individuals are infeasible, the winner is the one with a lower amount of constraints violation.

This approach does not need any extra parameter for the constraint handling mechanism, but it requires niching [40] to maintain diversity. From here, we will refer to this technique as TS, because its key feature is its tournament selection mechanism.

Hinterding and Michalewicz proposed a method with mate restrictions [65]. Before the recombination, the parents are chosen in such a way that they complement each other. The first parent is chosen considering only its objective function value and its feasibility, while the second is chosen seeking for the minimum of satisfied constraints shared with the first parent. This mechanism was designed with the aim to produce better descendants, recombining complementary parents according to the violated constraints.

Another technique which makes a separation of objectives and constraints, is the extension made by Schoenauer and Xanthakis [158] to the approach called behavioral memory, to make it able to handle problems with constraints. This technique performs one optimization phase per constraint, optimizing the constraint itself; later on, and assuming that the previous phases produced enough feasible solutions in the population, the approach proceeds to the optimization of the objective function. This approach is normally very sensitive to the order in which the constraints are optimized.

The use of multiobjective concepts for constrained optimization is part of this category. The original constraints of the problem can be seen as objectives, where we try to minimize the violation. Such objectives can be added to the original set of objective functions, and a multiobjective approach can be applied then.

The first example of these techniques is the COMOGA by Surry et al. [167, 166], which adopts the multiobjective approach called VEGA [156] and Pareto ranking. Parmee and Purchase [131] also apply VEGA in a real world problem. Camponogara and Talukdar [11] transform the constrained optimization problem into a bi-objective problem, where the first objective is the original one, and the second is the sum of violations. Jiménez and Verdegay [78] use the min-max method, together with selection rules very similar to the rules of Deb [38] previously mentioned.

An approach based on sub-population, similar to VEGA, was proposed

by Coello Coello [19]. In each sub-population the objective is to minimize the violation of one constraint. In a further approach [18], the author proposes the use of Pareto ranking to minimize violations, and, at the same time, optimize the objective function. Coello Coello and Mezura-Montes [22] proposed another approach based on multiobjective concepts, in which the selection mechanism consists on tournaments based on dominance.

Ray et al. [141] proposed a technique in which the ranking takes place considering the objective function value and the violation of constraints of the individuals. In addition to this ranking, the approach performs a mechanism of complement of constraints, similar to the approach of Hinterding and Michalewicz [65].

Runarsson and Yao proposed the technique called stochastic ranking (SR) [153], in which the aim is to balance the influence of the objective function and the penalty function when assigning fitness to a solution. SR does not require the definition of a penalty factor. Instead, a user-defined parameter called P_f sets the probability of using only the objective function to compare two solutions to sort them. The selection process is based on a ranking process. Then, when the solutions are sorted using a bubble sort-like algorithm, sometimes, depending of the P_f value, the comparison between two adjacent solutions will be performed using only the objective function. This algorithm is based on an evolution strategy with global intermediate recombination, applied only to the strategy parameters (not to the decision variables of the problem). For the validation of this approach, the authors adopted the test suite of Michalewicz and Schoenauer [116], and the results were equal or better than those of the homomorphous mappings [90], requiring a lower computational cost. Because of the excellent results that this approach produces, stochastic ranking became another reference point when comparing evolutionary algorithms for constrained optimization.

3.1.5 *Hybrid methods*

Hybrid methods are those with two or more different algorithms for optimization. They can be mathematical programming methods or metaheuristics.

One example of hybridization with a mathematical programming method is the approach of Adeli and Cheng [1]. They couple a genetic

algorithm with the method of the augmented lagrangian for a penalty function. Kim and Myung [84, 121] proposed an approach of two phases, where the first phase is based on Lagrangian penalties.

The CORE (Constrained Optimization by Random Evolution) approach [6] adopts the Nelder-Mead simplex method [122] for optimization without constraints, together with random evolutionary search to minimize the violation of constraints.

A fuzzy logic hybrid was proposed by Van Le [102], where the original constraints are replaced by *fuzzy constraints*, defined in the same work. The search engine of this approach is evolutionary programming [51].

The artificial immune system has been also used for constrained optimization. The immune systems have a population of antigens, and another population of antibodies. While the antigens are undesirable agents, the antibodies are agents which adapt themselves, in order to obtain a resemblance to the antigens, which are then able to neutralize them. The first proposal to handle constraints with an artificial immune system, was made by Hajela and Lee [60, 61], where the antigens were a set of feasible individuals, and a genetic algorithm is used to do the actual optimization (the artificial immune system is only used to move solutions towards the feasible region). The expression strategies [63] are a refinement of the previous proposal based on an artificial immune system. Coello Coello and Cruz Cortés [17] proposed improvements to Hajela and Lee's proposal, both in a serial and a parallel implementation. Another version of an artificial immune system for constrained optimization which is not hybridized with any other metaheuristic, was proposed by Cruz Cortés et al. in [32].

Bilchev and Parmee [8, 9] proposed an ant colony optimization-based approach for constrained optimization. The ant colony algorithm [42] was inspired by the behavior of ants while searching for food. This approach for handling constraints defines some sources of food as unacceptable, when they violate constraints.

Another metaheuristic that has become very popular for constrained optimization, is particle swarm optimization [83], which is inspired on the movements of a group of animals, like birds or fish. An early approach based on particle swarm is the one reported in [133], which uses a dynamic penalty function. Hu el al. [69] also proposed an approach for constrained optimization. Toscano-Pulido and Coello Coello [169] proposed a particle swarm algorithm with *turbulence* (a mutation-like operator); the technique performs comparisons between particles through a set of rules, similar

to those of [38], in order to choose the best particles of the swarm. The approach of Muñoz-Zavala et al. [123] adopts a set of rules very similar to the previous approach, but implements *perturbation* (an operator similar to the one used by the differential evolution algorithm).

3.2 Cultural algorithms in constrained and Real-Valued optimization problems

Cultural algorithms are considered hybrid methods, but in this thesis will be described in a separate section because of their relevance for this work.

Reynolds et al. [148] and Chung and Reynolds [15] have explored the use of cultural algorithms for global optimization with very encouraging results. The first approach is a modification of the GENOCOP approach [114] to incorporate it a belief space, but the technique only deals with linear constraints, as the original GENOCOP. The second approach is an evolutionary programming-based algorithm, with the same belief space adopted for GENOCOP.

The belief space of these approaches is based on the *interval-schemata* of Eshelman and Schaffer [46]. The belief space represents varying numerical intervals, one for each decision variable. In this case, those intervals are adapted to reflect the feasible region of the problem, with the information acquired by the individuals of previous generations. At the beginning, the intervals represent the whole search space, but they are tightened to represent the known feasible region. With this structure as the belief space, only convex feasible regions can be represented.

Although the belief space is similar for those approaches, the influence function must be different for each population space, due to the fact that each of them has its own variation operators:

- As we said before, GENOCOP's crossover operator is a linear combination of the parents. Now, one of the parents must be inside the intervals stored in the belief space. Mutation operators are also modified, in order to move the individuals nearer to the region defined by the belief space.

- Because evolutionary programming only performs mutation, this operator was modified to generate new individuals nearer to the region

defined by the belief space, and then spread along these intervals. Besides mutation, a repair operator was also implemented, to transform infeasible solutions into feasible ones.

3.2.1 *CAEP*

Based on their previous work, Chung and Reynolds [16] use evolutionary programming with a mutation operator influenced by the best individual found so far, and the intervals where good solutions have been found. They call their approach *CAEP*, or Cultural Algorithms with Evolutionary Programming. They follow their work with evolutionary programming, probably because of the low computation cost it presented when compared to the genetic algorithm of GENOCOP [15].

The pseudo code of a CAEP is shown in Algorithm 6 [14]. The selection mechanism is the same adopted in the standard evolutionary programming, conformed by stochastic tournaments.

Algorithm 6: Cultural algorithm with evolutionary programming

```
initialize(P = {x_1, ..., x_μ})
evaluate(P)
initialize(B)
repeat
  for j = 1 to μ do
    for i = 1 to n do
      x'_{i,j} = mutation(x_{i,j}, influence(B))
    end for
  end for
  evaluate(P')
  P = selection(P ∪ P')
  update(B, accept(P))
until the termination condition is achieved
```

A component of the belief space, based on the interval-schemata, was called *normative knowledge*, because it states the norms that most of the individuals must follow (to be socially accepted, according to the cultural evolution model).

Besides normative knowledge, another knowledge source was added to the belief space of the CAEP, and that is the *situational knowledge*, which stores the best individual(s) found so far. This source presents the best individuals as leaders to follow. The name is because this knowledge source represents the specific situation of the best individuals, which is attractive to others.

Chung performed experiments with a CAEP with situational knowledge, with normative knowledge, or with both sources [14]. However, Chung's approaches are designed for unconstrained optimization.

Jin and Reynolds [79] proposed a CAEP for constrained optimization. Their main addition is an n-dimensional regional-based schema, called *belief-cell* (renamed later as *topographical knowledge*), as an explicit mechanism that supports the acquisition, storage and integration of knowledge about nonlinear constraints in a cultural algorithm.

The cells are labeled, based on the points generated within them, as feasible, infeasible or semifeasible. The idea of this approach is to build a map of the search space which is used to derive rules about how to guide the search of the CAEP (avoiding infeasible regions and promoting the exploration of feasible and semi-feasible regions). As the cells are independent, this map is able to represent non-convex, or even disjoint feasible regions.

Besides the topographical knowledge, the authors only adopted the normative knowledge (they removed the situational source).

Later on, Jin and Reynolds adopted the previously proposed algorithm for applications in data mining [80]. In this new version, the idea of using a data structure for the topographical knowledge appears, because a static set of cells involves a heavy memory usage. As indicated before, this work focuses on data mining applications, and not on constrained optimization.

Using the same population space (evolutionary programming), Saleem and Reynolds proposed a cultural algorithm for dealing with dynamic environments [154], which adds two more ways to incorporate domain knowledge to CAEPs, in addition to the existing proposals by Chung and Jin. These knowledge sources are *history knowledge* and *domain knowledge*, and are designed to extract patterns in environmental changes. The history knowledge extracts patterns in the changes of optima's location, when environment changes occur, in order to predict new changes. The domain knowledge is specialized information about the problem at hand; for example, Saleem adopts the cones world [120], and knowledge about the fitness landscape is encoded in this source. The domain knowledge

source is somewhat difficult to implement when dealing with a broad class of problems.

Using as a basis the work of Jin and Reynolds, a CAEP for constrained optimization was developed in [26, 23], where a spatial data structure is incorporated to store the map of the feasible region (topographical knowledge), and also new rules are used in several phases of the algorithm. This approach is the immediate antecedent, and the main motivation for this thesis work.

3.2.2 *Other search engines for cultural algorithms*

In [73, 72], Iacoban et al. change the evolutionary programming algorithm of the population space for a particle swarm optimizer [83], with the aim to enhance convergence. They make an analysis of the effects of the belief space over the evolutionary process, showing the similarities with CAEP, and identifying the phases of the search process with the help of the belief space.

Peng and Reynolds returned to evolutionary programming in [134], to perform an analysis similar to the one from [73], about the roles of the knowledge sources. Both works were developed for unconstrained optimization.

Another approach based on particle swarm was developed in [43], this time using only normative knowledge. The authors also explored a cultured version of the gaussian particle swarm optimization (a modification to the uniform distributions commonly applied in particle swarm optimization).

These examples show the necessity to explore new search engines for the population space in cultural algorithms, in order to improve their results. In this work we use, for the first time, differential evolution in a cultural algorithm. Note however that differential evolution has been applied to contrained optimizaton several times in the past, as we will see in the following section.

3.3 Differential evolution in constrained optimization

The number of approaches based on differential evolution that have been proposed for constrained optimization has considerably increased in the last few years. We will review next the most representative of them.

One of the original developers of differential evolution, Storn, proposed constraint adaptation [165], in which all the constraints of the problem at hand are relaxed, so that all the individuals in the initial population become feasible. The constraints are reduced toward their original versions at each generation, but the individuals must always remain feasible (*i.e.*, different relaxations are applied at each generation). The author says that this approach is not suitable for handling equality constraints, and one of its main applications is constraint satisfaction (where only constraint violation is important, and there is no objective function).

Another constraint handling technique is the one proposed by Lampinen [94] (we will refer to this technique as DE). He states some rules for the replacement made during the selection procedure, that can be summarized as follows:

- If both individuals are feasible, the one with a better value of the objective function always wins.
- If the newly generated individual is feasible, as his parent is infeasible, the new individual is used for the next generation.
- If both individuals are infeasible, the parent is replaced if the new individual has lower or equal violation for all the constraints (this comparison is made in the Pareto sense in the constraint violation space).

These rules are very similar to those of Deb's approach TS [38], but the main difference between them is the comparison for the case of two infeasible individuals: while Lampinen makes a comparison in the Pareto sense, Deb sums all the constraint violations and compares a single value.

The rest of the differential evolution algorithm remains the same. The experiments are done with 10 of the 11 test functions proposed in [90]. Some previous versions of this algorithm appeared in [93, 92], where the

replacement rules were not as complete as in [94], and less test problems were taken into account.

Simultaneously to Lampinen, Lin et al. [105] proposed the use of an augmented Lagrangian function to guide the search, with a newly developed method to update the multipliers. In a first phase, the multipliers are constant and the problem is minimized. During a second phase, the multipliers are updated and the algorithm tries to maximize the dual function. Lin et al. [105] called their approach hybrid differential evolution, because they add some new steps to the original algorithm. Such steps are acceleration and migration, and are used when the current population has either too much or no diversity.

Mezura-Montes et al. [112, 111] have experimented with the generation of several children per parent, which compete among them to define the one that will survive in a local tournament. They also propose a diversity mechanism which gives infeasible point some probability to survive.

4

Proposed Approach for Constrained Optimization

In this chapter we will describe the approach proposed in this thesis for solving constrained optimization problems.

Our proposed approach uses differential evolution in the population space. In previous work, we performed some experiments with a CAEP [22], but our results although encouraging at first, could not be improved in spite of our efforts. While looking for a more robust search engine for our cultural algorithm, we came across differential evolution, which is a heuristic that is specifically designed for global optimization of real-valued functions, and it presents a highly competitive performance in unconstrained problems even without a careful fine-tuning (as normally required by other evolutionary algorithms) [164]. After performing some preliminaty experiments, it became evident that differential evolution was a much better choice for the purposes of this thesis, and thus we decided to adopt it as our search engine.

A pseudocode of our proposed approach (called cultured differential evolution, or CDE) is shown in Algorithm 7.

In the initial steps of the algorithm, a population of μ individuals is created, as well as a belief space. For the offspring generation, the variation operator of the differential evolution algorithm is influenced by the belief space.

Algorithm 7: Cultured differential evolution

```
Generate initial population
Evaluate initial population
Initialize the belief space
repeat
  for each individual in the population do
    Apply the variation operator influenced by a randomly chosen
    knowledge source
    Evaluate the child generated
    Replace the individual with the child, if the child is better
  end for
  Update the belief space with the accepted individuals
until the termination condition is achieved
```

4.1 Constraint-handling mechanism

Since we want to solve constrained optimization problems, the objective function by itself does not provide enough information as to guide the search properly. To determine if a child is better than its parent, and, therefore, it can replace it, we use the following rules in a binary tournament selection scheme:

1. A feasible individual is always better than an infeasible one.
2. If both are feasible, the individual with the best objective function value is better.
3. If both are infeasible, the individual with less amount of constraint violation is better.

The amount of constraint violation is measured with normalized constraints, adopting the following expression:

$$viol(\mathbf{x}) = \frac{1}{\ell}\sum_{k=1}^{\ell}\frac{g_k(\mathbf{x})}{\max(g_k)}$$

where $g_k(\mathbf{x})$ is the k-th of ℓ constraints of the problem, and $\max(g_k)$ is the largest violation of the constraint g_k found so far.

These tournament rules were based on other works reported in the specialized literature ([94, 38]). When comparing infeasible individuals, the proposed approach is more similar to Deb's approach, since Lampinen's approach makes the comparison of constraint violations in the Pareto sense. However, the above expression describes the normalization of the constraints; this is done in order to allow a fair comparison among constraint values, regardless of the units in which each constraint is expressed[1]. Such a normalization was not adopted in Deb's approach. Also, our approach does not use niching to maintain diversity, as in Deb's proposal [38].

A shared feature of the three approaches is the fact that the evaluation of the objective function is only needed when comparing two feasible individuals, as can be seen from the tournament rules above. This reduces the amount of CPU time required when solving highly constrained problems.

The rest of the algorithm, described below, is rather different from the previous approaches.

4.2 The belief space

In our CDE, the belief space is divided into four knowledge sources. Each knowledge source helps in different stages of the optimization process, and for different problems. They are described next.

4.2.1 *Situational knowledge*

The situational knowledge consists of the best exemplar $\mathbf{e}_{\text{all}}$ found along the evolutionary process. It represents a leader for the other individuals to follow, and gives the algorithm the stronger component of exploitation. This knowlegde source is useful mainly during the final stages of the optimization process, when the algorithm refines the current solution.

Initialization

To initialize the situational knowledge, it is necessary to have an initial population, so that we can find the best individual and store it.

[1]This sort of situation about the units of different functions is called "the incommensurability problem".

Influence

The variation operators of differential evolution are influenced in the following way:

$$x_{i,j}^{'} = e_{i,\text{all}} + F(x_{i,r1} - x_{i,r2})$$

where $e_{i,\text{all}}$ is the i-th component of the individual stored in the situational knowledge. This operator uses the leader instead of a randomly chosen individual for the recombination. This has the effect of pushing the newly generated individuals closer to the best point found.

This way of influencing the variation operator was previously proposed in a variant of the differential evolution algorithm [164], and is called DE/best/1/bin (the rest of the mechanisms are the same as in the DE/rand/1/bin variant; indeed, the word "rand" or "best" indicates if the individual $r3$ is chosen at random or is the best of the population,). The difference with that previous proposal is that we use several modifications of the variation operator, and not only one.

Update

The update of the situational knowledge is done by replacing the stored individual, $\mathbf{e}$, by the best individual found in the current population, $\mathbf{x}_{\text{best}}$, only if $\mathbf{x}_{\text{best}}$ is better than $\mathbf{e}_{\text{all}}$.

$$\mathbf{e}_{\text{all}} = \begin{cases} \mathbf{x}_{\text{best}} & \text{if } \mathbf{x}_{\text{best}} \text{ is better than } \mathbf{e}_{\text{all}} \\ \mathbf{e}_{\text{all}} & \text{otherwise} \end{cases}$$

4.2.2 *Normative knowledge*

The normative knowledge contains the intervals for the decision variables where good solutions have been found, in order to move new solutions towards those intervals. This knowledge source also exploits some regions, but the mechanism is not so aggressive as the one of situational knowledge. In some sense, it also explores, but only the region of good solutions (which is the whole search space at the begining of the algorithm execution).

The normative knowledge has the structure shown in Figure 4.1.

In Figure 4.1, l_i and u_i are the lower and upper bounds, respectively, for the i-th decision variable, and L_i and U_i are the values of the fitness

l_1	u_1	l_2	u_2	$\cdots$	l_n	u_n
L_1	U_1	L_2	U_2	$\cdots$	L_n	U_n

dm_1	dm_2	$\ldots$	dm_n

Figure 4.1: Structure of the normative knowledge

function associated with that bound. Also, the normative knowledge includes a scaling factor, dm_i, to influence the mutation operator adopted in differential evolution.

Initialization

To initialize the normative knowledge, all the bounds are set to the intervals given as input data of the problem. L_i and U_i are set to $+\infty$, assuming a minimization problem, and $dm_i = u_i - l_i$, for $i = 1, 2, \ldots, n$, to have a null influence at the first generation.

Influence

The following expression shows the influence of the normative knowledge on the variation operators:

$$x'_{i,j} = \begin{cases} x_{i,r3} + F\,|x_{i,r1} - x_{i,r2}| & \text{if } x_{i,r3} < l_i \\ x_{i,r3} - F\,|x_{i,r1} - x_{i,r2}| & \text{if } x_{i,r3} > u_i \\ x_{i,r3} + F\left(\frac{u_i - l_i}{dm_i}\right)(x_{i,r1} - x_{i,r2}) & \text{otherwise} \end{cases}$$

We introduce the scaling factor $\frac{u_i - l_i}{dm_i}$ for the mutation to be proportional to the interval of the normative knowledge for the i-th decision variable. This scaling factor is adopted to spread the solutions within all the promising intervals, which introduces some exploration (only on the stored intervals). This mechanism of scaling the differential operator can be seen as a way to self-adapt the value of the parameter F of the differential evolution algorithm.

Update

The update of the normative knowledge is as follows: let $A = \{a_1, \ldots, a_{\text{accepted}}\}$ be the set of indices of the accepted individuals in the population, and

$$amin_i \in A \mid x_{i,amin_i} = \min(x_{i,a_1}, \ldots, x_{i,a_{\text{accepted}}})$$

and

$$amax_i \in A \mid x_{i,amax_i} = \max(x_{i,a_1}, \ldots, x_{i,a_{\text{accepted}}})$$

be the indices of the individuals with minimum and maximum values, respectively, for the parameter i. These minumum and maximum values are obtained only among the accepted individuals. Then, the intervals are updated using the following expressions:

$$l_i = \begin{cases} x_{i,amin_i} & \text{if } x_{i,amin_i} < l_i \vee f(\mathbf{x}_{amin_i}) < L_i \\ l_i & \text{otherwise} \end{cases}$$

$$u_i = \begin{cases} x_{i,amin_i} & \text{if } x_{i,amin_i} > u_i \vee f(\mathbf{x}_{amin_i}) < U_i \\ u_i & \text{otherwise} \end{cases}$$

In words, the update will reduce or expand the intervals stored on normative knowledge. An expansion takes place when the accepted individuals do not fit in the current interval, while a reduction occurs when all the accepted individuals lie inside the current interval, and the extreme values have a better fitness and are feasible.

If the values of l_i or u_i are updated, the same must be done with L_i or U_i.

The dm_i values are updated with the greatest difference $|x_{i,r1} - x_{i,r2}|$ found during the application of the variation operators at the previous generation.

4.2.3 *Topographical knowledge*

The usefulness of the topographical knowledge is to create a map of the problem fitness landscape during the evolutionary process. It also identifies good regions, but they can be disjoint and spread throughout the entire search space.

It consists of a set of cells, and the best individual found on each cell. The topographical knowledge, also, has an ordered list of the best b cells, based on the fitness value of the best individual on each of them. For the sake of a more efficient memory management, in the presence of high dimensionality (i.e., too many decision variables), we use an spatial data structure, called k-d tree, or k-dimensional binary tree [7]. In k-d trees, each node can only have two children (or none, if it is a leaf node), and represents a division in half for any of the k dimensions. In Figure 4.2, we show an example of a k-d tree representing a two-dimensional space; every no-leaf node represents a region that is divided in half, while leaf nodes represent a not divided region.

Initialization

To initialize the topographical knowledge, we only create the root node, which represents the entire search space, and contains the best solution found in the initial population.

Influence

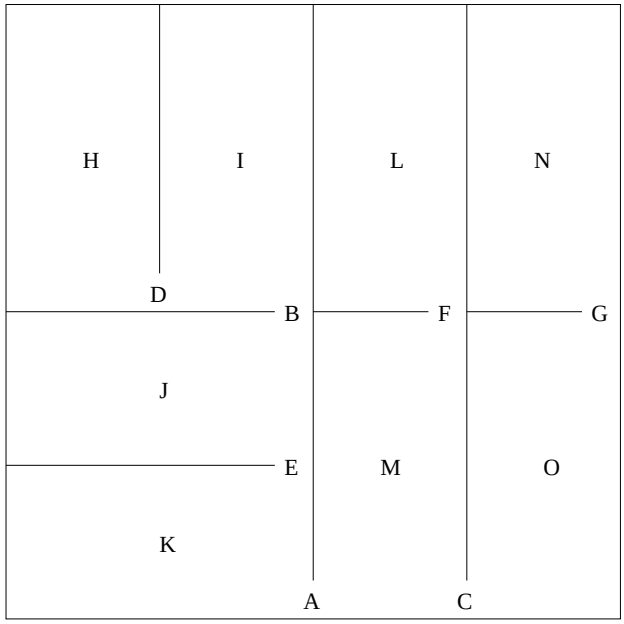
The influence function tries to move the children to any of the b cells in the list:

$$x_{i,j}^{'} = \begin{cases} x_{i,r3} + F\,|x_{i,r1} - x_{i,r2}| & \text{if } x_{i,r3} < l_{i.c} \\ x_{i,r3} - F\,|x_{i,r1} - x_{i,r2}| & \text{if } x_{i,r3} > u_{i,c} \\ x_{i,r3} + F(x_{i,r1} - x_{i,r2}) & \text{otherwise} \end{cases}$$

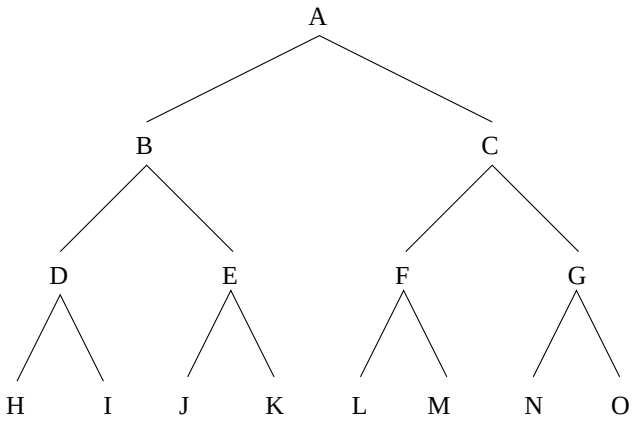
where $l_{i,c}$ and $u_{i,c}$ are the lower and upper bounds of the cell c for the parameter i, such cell is randomly chosen from the list of the b best cells.

Update

The update function splits a node if a better solution is found in that cell, and if the tree has not reached its maximum depth (defined as an input parameter of the approach). The dimension in which the division is done, is the one that has a greater difference between the solution stored and the new reference solution (i.e., the new solution considered as the "best" found so far).



(a) Space represented



(b) Corresponding tree

Figure 4.2: Example of the partition of a two-dimensional space by a k-d tree

$\mathbf{e}_1$	$\mathbf{e}_2$	$\cdots$	$\mathbf{e}_w$

ds_1	ds_2	$\cdots$	ds_n
dr_1	dr_2	$\cdots$	dr_n

Figure 4.3: Structure of the history knowledge

4.2.4 *History knowledge*

This knowledge source was originally proposed for dynamic objective functions, and it was used to find patterns in the environmental changes [154]. History knowledge records in a list, the location of the best individual found before each environmental change. That list has a maximum size w.

The structure of the history knowledge is shown in Figure 4.3, where $\mathbf{e}_j$ is the best individual found before the j-th environmental change, ds_i is the average distance of the changes for parameter i, and dr_i is the average direction if there are changes for parameter i. In our approach, instead of detecting changes of the environment, we store a solution if it remains as the best one during the last p generations. If this happens, we assume that we are trapped in a local optimum. Thus, this knowledge source is helpful to escape from local optima, and has a limited ability to predict new local (and possibly global) optima. We also use this mechanism to introduce diversity to the search, as it can be seen from its influence described next.

Initialization

The initialization of this knowledge source consists of an empty list of previous local optima. The ds_i values are set to dm_i, and the dr_i values are randomly set to 1 or -1; these values are chosen with the aim of making its influence more similar to the standard neutral operator.

Influence

The expression of the influence function of the history knowledge is the following:

$$x_{i,j}^{'} = \begin{cases} e_{i,1} + F \cdot dr_i \cdot |x_{i,r1} - x_{i,r2}| & \text{if } U(0,1) < \alpha \\ e_{i,1} + \frac{ds_i}{dm_i} \cdot (x_{i,r1} - x_{i,r2}) & \text{if } U(0,1) < \beta \\ U(lb_i, ub_i) & \text{otherwise} \end{cases}$$

where $e_{i,1}$ is the i-th decision variable of the previous best $\mathbf{e}_1$ stored in the list of the history knowledge; dm_i is the maximum difference for the i-th variable, stored in the normative knowledge; lb_i and ub_i are the lower and upper bounds of the i-th variable of the problem, given as input; and $U(a, b)$ is a realization of a random variable with uniform distribution between a and b.

The first part of the equation changes the direction of the operator, while the second part changes its magnitude. The third part is for diversity addition to the algorithm.

Update

To update the history knowledge, we add to the list any local optima found during the evolutionary process. If the list has reached its maximum length w, the oldest element is discarded. The average distances and directions of change are calculated by:

$$ds_i = \frac{\sum_{j=1}^{w-1} |e_{i,j+1} - e_{i,j}|}{w - 1}$$

$$dr_i = \operatorname{sgn}\left(\sum_{j=1}^{w-1} \operatorname{sgn}\left(e_{i,j+1} - e_{i,j}\right)\right)$$

where the function sgn returns the sign of its argument. Through these expresions, we obtain the average changes of magnitude and direction, respectively, of each parameter for the previous local optima.

4.3 Acceptance function

The number of individuals accepted for the update of the belief space is computed according the design of a dynamic acceptance function pro-

posed by Saleem [154]. The number of accepted individuals decreases as the generation number increases.

Saleem [154] suggests to reset the number of accepted individuals when an environmental change occurs. In our case, we reset the number of accepted individuals when the best solution has not changed in the last p generations.

We obtain the number of accepted individuals, $|A|$, with the following expression:

$$|A| = \left\lfloor \mu p_{\text{accept}} + \frac{\mu(1 - p_{\text{accept}})}{g} \right\rfloor$$

where p_{accept} is a parameter given by the user, within the range $(0, 1]$; Saleem [154] suggests using 0.2, noticing the parallelism of this paramenter with the $1/5$ success rule of Rechenberg. g is the generation counter, but is reset to 1 when the best solution has not changed in the last p generations.

4.4 Main influence function

The main influence function is responsible for choosing the knowledge source to be applied to the variation operator of differential evolution. At the beginning, all the knowledge sources have the same probability to be applied, $r_{ks} = \frac{1}{4}$, because there are 4 knowledge sources; but during the evolutionary process, the probability of the knowledge source ks to be applied is:

$$r_{ks} = 0.1 + 0.6\frac{v_{ks}}{v}$$

where v_{ks} is the number of times that an individual generated by the knowledge source ks outperforms its parent in the current generation, and v is the number of times that an individual generated (by any knowledge source) outperforms its parent in the current generation. The lower bound of r_{ks} is set arbitrarily to 0.1, to ensure that any knowledge source has always a probability greater than 0 to be applied. If $v = 0$ during a generation, then $r_{ks} = \frac{1}{4}$, as in the beginning.

Such are the main mechanisms of the technique. Preliminary stages of this algorithm were published in [96, 95]. The final version described here was published in [98], and a journal version was published in [97].

4.5 Parameters of the technique

The proposed approach has several parameters that the user must set by hand in order to successfully use the algorithm. The following is a list of all the parameters and some suggestions about their settings.

- Population size, μ, is the number of individuals in the population. A value as small as 10 gives good results in "easy" (i.e., with few decision variables and constraints easy to satisfy) problems. However, in general, we recommend to adopt a population size of 100. Larger sizes may be adopted, but such values are recommended only for very hard problems if one can afford the extra computational cost.

- Maximum number of generations, $g_{\max}$. Is the number of iterations that the algorithm will run. This parameter, together with the population size, defines the number of fitness function evaluations that the approach will perform. For easy problems, one may start with 100 or 200 generations. Then, if necessary, this value can get increased until no improvement in the results is obtained. In general, and based on our own experience, we suggest to set this parameter to 1,000.

- F and CR are the parameters of differential evolution. These parameters can be set following the suggestions in [138]. Good default values are $F = 0.5$ and $CR = 1$.

- Maximum depth of the k-d tree. The larger this value, the more accurate the map generated by the topographical knowledge source will be, but more memory will be needed. If m is this maximum depth, the tree can have up to $2^m - 1$ nodes. This parameter depends of the free memory available in the device in which the algorithm will run. With a maximum depth of 12, the algorithm can have up to 4095 nodes, which is a manageable value for most current computers. So, we suggest to set this parameter to 12.

- The length b of the best cells list is the number of independent cells that will be considered for the topographical knowledge. We suggest to adopt a number of independent cells equal to the number of decision variables of the problem.

- The size of the list in the history knowledge, w, is the number of previous local optima that will be considered when looking for the next one. This parameter has little influence on the performance of the algorithm when the distribution of the local optima is not regular in the problem. We recommend to set this parameter to 5.

- α and β have little influence on performance, and can be fixed to 0.4 or 0.45.

- p_{accept} is the percentage of accepted individuals at the end of the evolutionary process. Saleem [154] suggests using 0.2, which is the value that we adopted. However, this value must be increased if the algorithm exhibits premature convergence.

5

Results for the Constrained Optimization Approach

5.1 Standard problems

To validate our approach, we adopted the well-known benchmark originally proposed in [116] and extended in [153] which has been often used in the literature to validate new evolutionary constraint-handling techniques. It contains several problems with different characteristicis: large and small fesible regions, linear and non linear objective functions, linear and non linear constraints, equality and inequality constraints, etc.

The expressions of the 13 test problems are presented in Appendix A.

5.1.1 *Comparison of results*

The parameters used by our approach are the following: $\mu = 100$, $g_{\max} = 1000$, the factors of differential evolution are $F = 0.5$ and $CR = 1$, maximum depth of the k-d tree = 12, length of the best cells list $b = 10$, the size of the list in the history knowledge $w = 5$, $\alpha = \beta = 0.45$, and $p_{\text{accept}} = 0.2$. These parameters were empirically derived after numerous experiments. This parameter setting was found to be a good compromise for all the test functions; optimal settings for each problem may exist, but they are not reported here. It is worth noticing that we did not spend too much effort in performing a very thorough parameters fine-tuning, since we found that

our approach was relatively robust. However, it is expected that a better performance (even if only marginal in some cases) may be obtained with different settings. For each test function, we performed 30 independent runs. The results for the 13 problems are shown in Table 5.1.

We compare our approach to five state-of-the-art approaches: the Homomorphous Mappings (HM) [90], Stochastic Ranking (SR) [153], the Adaptive Segregational Constraint Handling Evolutionary Algorithm (ASCHEA) [64], a Constraint Handling Method for Genetic Algorithms (TS) [38], and a Constraint Handling Approach for Differential Evolution (DE) [94]. All these approaches were briefly described in Chapter 3.

The best results obtained by each approach are shown in Table 5.2. The mean values provided are compared in Table 5.3 and the worst results are presented in Table 5.4. The results provided by these approaches were taken from the original references for each method.

The results of HM were obtained with **1,400,000** evaluations of the fitness function, the results of SR required **350,000** evaluations, and the results of the ASCHEA technique were obtained with **1,500,000** evaluations of the fitness function. The results reported for TS and DE required a variable number of fitness function evaluations and used different parameter settings for each problem. TS required from **250,050 to 350,100** function evaluations (in the original source, the number of evaluations and maximum generations for the problem g01 are missing), and DE required from **10,000 to 12,000,000** function evaluations. Our approach required **100,100** evaluations in all the test problems adopted.

The best result obtained by CDE in problem g01 is shown in Table 5.5. The cultured differential evolution, SR and DE reached the optimum value in all the 30 runs performed. TS reached the optimum in its best and median case, ASCHEA reached the optimum only in its best case, and HM could not reach the optimum.

For the problem g02, SR has a lower variability than the cultured differential evolution. However, our approach can get the optimum value in some of the runs, and HM, SR and ASCHEA were unable to do it. The best value obtained is shown in Table 5.6.

In g03, the best result obtained by the cultured differential evolution is shown in Table 5.7, which is very close to the global optimum. SR and DE are clear winners in this problem, because they were able to reach the optimum in all the runs reported. HM and ASCHEA also had a better performance than the cultured differential evolution in this problem.

Problem	Optimal	Results of the cultured differential evolution algorithm			
		Best	**Mean**	**Worst**	**St. Dev.**
g01	**-15**	**-15.000000**	-14.999996	-14.999993	0.000002
g02	**0.803619**	**0.803619**	0.724886	0.590908	0.070125
g03	**1**	0.995413	0.788635	0.639920	0.115214
g04	**-30665.539**	**-30665.538672**	**-30665.538672**	**-30665.538672**	**0.000000**
g05	**5126.4981**	5126.570923	5207.410651	5327.390497	69.225796
g06	**-6961.8138**	**-6961.813876**	**-6961.813876**	**-6961.813876**	**0.000000**
g07	**24.3062091**	**24.306209**	24.306210	24.306212	**0.000001**
g08	**0.095825**	**0.095825**	**0.095825**	**0.095825**	**0.000000**
g09	**680.6300573**	**680.630057**	**680.630057**	**680.630057**	**0.000000**
g10	**7049.25**	**7049.248058**	**7049.248266**	**7049.248480**	0.000167
g11	**0.75**	**0.749900**	0.757995	0.796455	0.017138
g12	**1**	**1.000000**	**1.000000**	**1.000000**	**0.000000**
g13	**0.0539498**	0.056180	0.288324	0.392100	0.167095

Table 5.1: Results obtained by our cultured differential evolution (CDE) approach

Problem	Optimal	Best Results of the compared techniques					
		CDE	HM	SR	ASCHEA	TS	DE
g01	-15	$\mathbf{-15.000000}$	-14.7864	-15.000	-15.0	-15.000	-15.000
g02	$\mathbf{0.803619}$	$\mathbf{0.803619}$	0.79953	0.803515	0.785	NA	NA
g03	$\mathbf{1}$	0.995413	0.9997	1.000	1.0	NA	1.0252
g04	$\mathbf{-30665.539}$	$\mathbf{-30665.538672}$	-30664.5	-30665.539	-30665.5	-30665.537	NA
g05	$\mathbf{5126.498}$	5126.570923	–	5126.497	5126.5	NA	5126.484
g06	$\mathbf{-6961.814}$	$\mathbf{-6961.813876}$	-6952.1	-6961.814	-6961.81	NA	-6961.814
g07	$\mathbf{24.306}$	$\mathbf{24.306209}$	24.620	24.307	24.3323	24.37248	24.306
g08	$\mathbf{0.095825}$	$\mathbf{0.095825}$	0.0958250	0.095825	0.095825	NA	0.095825
g09	$\mathbf{680.63}$	$\mathbf{680.630057}$	680.91	680.630	680.630	680.634460	680.630
g10	$\mathbf{7049.25}$	$\mathbf{7049.248058}$	7147.9	7054.316	7061.13	7060.221	7049.248
g11	$\mathbf{0.75}$	$\mathbf{0.749900}$	0.75	0.750	0.75	NA	0.74900
g12	$\mathbf{1.00}$	$\mathbf{1.000000}$	0.999999	1.000000	NA	NA	NA
g13	$\mathbf{0.053950}$	0.056180	NA	0.053957	NA	0.053950	NA

Table 5.2: Comparison of the best results of CDE with respect to HM [90], SR [153], ASCHEA [64], TS [38], and DE [94]. "-" means no feasible solutions were found. NA = Not Available. A result in **boldface** means that our approach obtained the same or a better value than any other of the techniques.

Problem	Optimal	Mean Results of the compared techniques					
		CDE	**HM**	**SR**	**ASCHEA**	**TS**	**DE**
g01	**−15**	−14.999996	−14.7082	−15.000	−14.84	−15.000	−15.000
g02	**0.803619**	0.724886	0.79671	0.781975	0.59	NA	NA
g03	**1**	0.788635	0.9989	1.000	0.99989	NA	1.0252
g04	**−30665.539**	**−30665.538672**	−30655.3	−30665.539	−30665.5	−30665.535	NA
g05	**5126.498**	5207.410651	–	5128.881	5141.65	NA	5126.484
g06	**−6961.814**	**−6961.813876**	−6342.6	−6875.940	−6961.81	NA	−6961.814
g07	**24.306**	**24.306210**	24.826	24.374	24.66	24.40940	24.306
g08	**0.095825**	**0.095825**	0.0891568	0.095825	0.095825	NA	0.095825
g09	**680.63**	**680.630057**	681.16	680.656	680.641	680.641724	680.630
g10	**7049.25**	**7049.248266**	8163.6	7559.192	7193.11	7220.026	7049.248
g11	**0.75**	0.757995	0.75	0.750	0.75	NA	0.74900
g12	**1.00**	**1.000000**	0.999134	1.000000	NA	NA	NA
g13	**0.053950**	0.288324	NA	0.067543	NA	0.241289	NA

Table 5.3: Comparison of the mean results of CDE with respect to HM [90], SR [153], ASCHEA [64], TS [38], and DE [94]. "-" means no feasible solutions were found. NA = Not Available. A result in **boldface** means that our approach obtained the same or a better value than any other of the techniques.

Problem	Optimal	Worst Results of the compared techniques					
		CDE	HM	SR	ASCHEA	TS	DE
g01	-15	-14.999993	-14.6154	-15.000	NA	-13.000	-15.000
g02	**0.803619**	0.590908	0.79119	0.726288	NA	NA	NA
g03	**1**	0.639920	0.9978	1.000	NA	NA	1.0252
g04	$\mathbf{-30665.539}$	$\mathbf{-30665.538672}$	-30645.9	-30665.539	NA	-29846.654	NA
g05	**5126.498**	5327.390497	–	5142.472	NA	NA	5126.484
g06	$\mathbf{-6961.814}$	$\mathbf{-6961.813876}$	-5473.9	-6350.262	NA	NA	-6961.814
g07	**24.306**	**24.306212**	25.069	24.642	NA	25.07530	24.307
g08	**0.095825**	**0.095825**	0.0291438	0.095825	NA	NA	0.095825
g09	**680.63**	**680.630057**	683.18	680.763	NA	680.650879	680.630
g10	**7049.25**	**7049.248480**	9659.3	8835.655	NA	10230.834	7049.248
g11	**0.75**	0.796455	0.75	0.750	NA	NA	0.74900
g12	**1.00**	**1.000000**	0.991950	1.000000	NA	NA	NA
g13	**0.053950**	0.392100	NA	0.216915	NA	0.507761	NA

Table 5.4: Comparison of the worst results of CDE with respect to HM [90], SR [153], ASCHEA [64], TS [38], and DE [94]. "-" means no feasible solutions were found. NA = Not Available. A result in **boldface** means that our approach obtained the same or a better value than any other of the techniques.

Design variable	Value
f	-15.000000
x_1	1.000000
x_2	1.000000
x_3	1.000000
x_4	1.000000
x_5	1.000000
x_6	1.000000
x_7	1.000000
x_8	1.000000
x_9	1.000000
x_{10}	3.000000
x_{11}	3.000000
x_{12}	3.000000
x_{13}	1.000000
g_1	0.000000
g_2	0.000000
g_3	0.000000
g_4	-5.000000
g_5	-5.000000
g_6	-5.000000
g_7	0.000000
g_8	0.000000
g_9	0.000000

Table 5.5: Best result obtained by CDE for g01

Design variable	Value
f	-0.803619
x_1	3.162681
x_2	3.128106
x_3	3.095032
x_4	3.061492
x_5	3.028413
x_6	2.994410
x_7	2.957990
x_8	2.921746
x_9	0.495094
x_{10}	0.488762
x_{11}	0.481956
x_{12}	0.476491
x_{13}	0.472845
x_{14}	0.465291
x_{15}	0.461291
x_{16}	0.457043
x_{17}	0.452091
x_{18}	0.448045
x_{19}	0.444050
x_{20}	0.440310
g_1	0.000000
g_2	-120.066861

Table 5.6: Best result obtained by CDE for g02

Design variable	Value
f	0.995413
x_1	0.304887
x_2	0.329917
x_3	0.319260
x_4	0.328069
x_5	0.326023
x_6	0.302707
x_7	0.305104
x_8	0.315312
x_9	0.322047
x_{10}	0.309009
g_1	0.000991

Table 5.7: Best result obtained by CDE for g03

The best result obtained for the function g04 is shown in Table 5.8. This problem is "easy" to solve for all the techniques, which present a similar behavior obtaining a value very close to the optimum in all the runs. The results for DE show a slightly different version of this problem, and therefore cannot be included in this comparison.

g05 is the problem in which the cultured differential evolution exhibits its highest variability of results, even when its best value is very close to the optimum (see Table 5.9). SR, ASCHEA and DE also obtained a best value very close to the optimum, but their variability is lower. HM was not able to find any feasible solution for this problem.

In Table 5.10 is shown the best result obtained for the cultured differential evolution in g06. In this problem, our approach shows a performance which is clearly better than that of HM and SR in terms of consistency to reach the optimum. Both had a large variability of results when solving this problem. ASCHEA and DE reported a mean (and worst in the case of DE) result very close to the optimum, as is the case of the cultured differential evolution.

The best result obtained by our approach in g07 is shown in Table 5.11. The cultured differential evolution exhibits a great robustness in this problem, reaching a value very close to the optimum in all the runs. The best

Design variable	Value
f	-30665.538672
x_1	78.000000
x_2	33.000000
x_3	29.995256
x_4	45.000000
x_5	36.775813
g_1	0.000000
g_2	-92.000000
g_3	-11.159500
g_4	-8.840500
g_5	-5.000000
g_6	-0.000000

Table 5.8: Best result obtained by CDE for g04

Design variable	Value
f	5126.570923
x_1	683.926335
x_2	1021.814124
x_3	0.116038
x_4	-0.397581
g_1	-0.036381
g_2	-1.063619
g_3	0.000803
g_4	0.000419
g_5	0.000980

Table 5.9: Best result obtained by CDE for g05

Design variable	Value
f	-6961.813876
x_1	14.095000
x_2	0.842961
g_1	0.000000
g_2	-0.000002

Table 5.10: Best result obtained by CDE for g06

value obtained for any of the other techniques (except DE), is worse than the worst result produced by the cultured differential evolution. Only DE can also reach the optimum, but exhibits a slightly larger variability (as can be seen in the worst case).

The cultured differential evolution approach obtained its best result shown in Table 5.12 for problem g08. This problem seems easy to solve for the techniques analyzed here, because all the techniques reached the optimum in almost all the runs, except HM, which exhibits a poor performance in this problem.

In problem g09, the best result obtained is shown in Table 5.13. SR and ASCHEA were able to reach the optimum in their best cases, while the algorithm proposed here and DE obtained the optimum in all the runs performed, being more robust.

In g10, the best result obtained by the cultured differential evolution is shown in Table 5.14. Again, our cultured differential evolution approach and DE obtained a value very close to the optimum in all the runs performed, being clear winners. However, DE required 270,000 function evaluations for this particular problem. All the other techniques exhibited a very high variability of results in this problem, and were not able to reach the optimum in their best cases.

The best result obtained for the function g11 is shown in Table 5.15. In this case, all the approaches analyzed here reached the optimum in their best cases. However, HM, SR, ASCHEA and DE were more robust in this problem, showing a low variability of results.

The best result obtained by our approach in g12 is shown in Table 5.16. This problem is another example of similar performance of SR and the cultured differential evolution, because they both could reach the optimum

Design variable	Value
f	24.306209
x_1	2.171982
x_2	2.363717
x_3	8.773931
x_4	5.095984
x_5	0.990654
x_6	1.430557
x_7	1.321617
x_8	9.828704
x_9	8.280092
x_{10}	8.376018
g_1	-0.000002
g_2	0.000000
g_3	0.000000
g_4	0.000000
g_5	-0.000010
g_6	-0.000002
g_7	-6.148630
g_8	-50.024352

Table 5.11: Best result obtained by CDE for g07

Design variable	Value
f	0.095825
x_1	1.227971
x_2	4.245373
g_1	-1.737460
g_2	-0.167763

Table 5.12: Best result obtained by CDE for g08

Design variable	Value
f	680.630057
x_1	2.330499
x_2	1.951372
x_3	-0.477541
x_4	4.365726
x_5	-0.624487
x_6	1.038131
x_7	1.594227
g_1	-0.000045
g_2	-252.561724
g_3	-144.878190
g_4	-0.000008

Table 5.13: Best result obtained by CDE for g09

Design variable	Value
f	7049.248058
x_1	579.380244
x_2	1359.992748
x_3	5109.875066
x_4	182.023843
x_5	295.604998
x_6	217.976157
x_7	286.418844
x_8	395.604998
g_1	0.000000
g_2	0.000000
g_3	0.000000
g_4	0.000000
g_5	-0.000537
g_6	-0.001600

Table 5.14: Best result obtained by CDE for g10

Design variable	Value
f	0.749900
x_1	0.707036
x_2	0.500000
g_1	0.000100

Table 5.15: Best result obtained by CDE for g11

Design variable	Value
f	1.000000
x_1	5.000000
x_2	5.000000
x_3	5.000000
g_1	-0.062500

Table 5.16: Best result obtained by CDE for g12

in practically all the runs performed.

Finally, in Table 5.17 is the best result obtained for the cultured differential evolution in g13. SR was more robust than our approach in this problem, and its best result was slightly better than ours.

In short, SR and DE are the most competitive constraint handling techniques compared here. However, our approach reached the global optimum in ten problems, while SR and DE did it in nine (DE was only tested in nine problems). Also, in most cases, the cultured differential evolution was more robust than SR, showing a very low standard deviation, while performing less than one third of its total number of fitness function evaluations. Finally, it is worth noticing that our approach used the same number of function evaluations and parameters for all the test functions, while DE got its parameters tuned for each problem, and requires a variable number of fitness function evaluations.

5.1.2 *Statistical analysis*

In order to determine the robustness of the approach proposed here, we performed a statistical analysis to obtain the confidence intervals of the

Design variable	Value
f	0.056180
x_1	-1.648857
x_2	1.515826
x_3	1.949583
x_4	0.753371
x_5	0.784312
g_1	0.000045
g_2	0.000839
g_3	0.000161

Table 5.17: Best result obtained by CDE for g13

mean statistic, for the 13 problems of the benchmark.

First, we need to identify the problems where at least two different solutions were provided in the 30 test runs performed. During the runs we performed, our approach always reached the optimum value in problems g04, g06, g08, g09 and g12. This means that we can have enough confidence that the algorithm will reach the optimum in these test problems. Thus, no statistical analysis will be done for them.

For the rest of the problems, we performed an Anderson-Darling goodness of fit test [163], which is a modification of the Kolmogorov-Smirnov test [160], for using informaton of a specific distribution when calculating critical values. We used a version of the Anderson-Darling test for the normal distribution.

The motivation for adopting this test is to ensure that the results of the algorithm do not have a normal distribution (alternative hypothesis). If the results come from a normal distribution (null hypothesis), we can easily obtain confidence intervals through a very simple expression. As it was expected, in every case, the data did not fit a normal distribution.

An alternative to obtain confidence intervals from unknown distributions is the bootstrap technique [160]. To obtain a bootstrap distribution for a statistic, we make "resamples" from the original sample, of the same size, choosing values at random with replacement. From each resample, we obtain the statistic of interest (the mean in this case), and construct a new sample with those values.

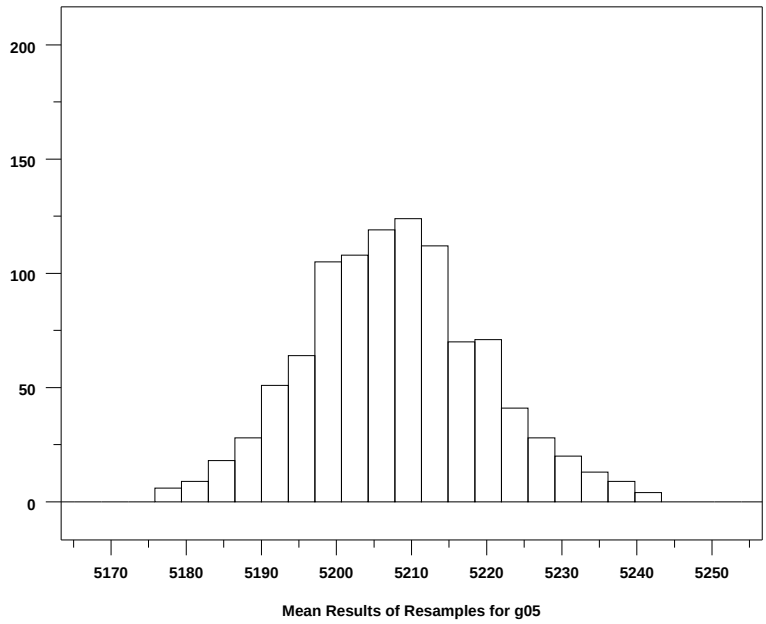


Figure 5.1: Bootstrap distribution for the mean statistic for problem g05.

In our case, we performed the experiments with 1,000 resamples for each problem. An example of the bootstrap distributions obtained is shown in the histogram from Figure 5.1. This histogram corresponds to the problem g05.

From the bootstrap distributions for each problem, we obtained the confidence intervals with the appropriate percentiles. The intervals were obtained with 95% of confidence. These results are sumarized in Table 5.18.

All the bootstrap distributions obtained are very close to the normal distribution. Actually, the Anderson-Darling test accepts all cases as data that come from a normal distribution, except in the cases where all the means are the same or very similar (g01, g07 and g10), and in the case of g13, because the bootstrap distribution is slightly skewed.

In any case, the bootstrap-percentile technique gives a good estimation of the confidence intervals. The intervals that we obtained are a good indicator of the robustness of the proposed approach. In five cases (g04, g06, g08, g09 and g12), the interval (for the precision used here) lies exactly within the best known solution. In three other cases (g01, g07 and g10),

Problem	Optimal	95% Confidence Interval on the Mean
g01	$\mathbf{-15}$	[-14.999996, -14.999997]
g02	$\mathbf{0.803619}$	[0.701130, 0.749364]
g03	$\mathbf{1}$	[0.749766, 0.827785]
g04	$\mathbf{-30665.539}$	**[-30665.538672, -30665.538672]**
g05	$\mathbf{5126.4981}$	[5183.8667, 5233.5552]
g06	$\mathbf{-6961.8138}$	**[-6961.813876, -6961.813876]**
g07	$\mathbf{24.3062091}$	[24.306210, 24.306210]
g08	$\mathbf{0.095825}$	**[0.095825, 0.095825]**
g09	$\mathbf{680.6300573}$	**[680.630057, 680.630057]**
g10	$\mathbf{7049.25}$	[7049.2476, 7049.2490]
g11	$\mathbf{0.75}$	[0.752695, 0.764359]
g12	$\mathbf{1}$	**[1.000000, 1.000000]**
g13	$\mathbf{0.0539498}$	[0.235337, 0.347311]

Table 5.18: Confidence intervals for the mean statistic.

the intervals are very small and very close to the optimum (the difference is only of one or two decimal places).

5.2 Engineering optimization problems

Additionally to this standard benchmark, we tested our proposed cultured differential evolution on an engineering optimization problem: design of trusses. Two different problems related to the design of trusses are tackled: the optimization of a 10-bar plane truss and a 200-bar plane truss [5], minimizing its weight, subject to displacement and stress constraints. The decision variables are the cross-sectional depth and width of each member of the truss. For further descriptions of these problems, please see the Appendix 3.

5.2.1 *Comparison of results*

The two engineering optimization problems previously mentioned were used by Belegundu [5] to evaluate the following numerical optimization techniques: Feasible directions (CONMIN and OPTDYN), Pshenichny's

Method	CONMIN	OPTDYN	LINRM	GRP-UI	SUMT
Weight	4793.0	9436.0	6151.0	5077.0	5070.0
Method	M-3	M-4	M-5	CDE	
Weight	4898.0	5057.0	5211.0	**4656.39**	

Table 5.19: Results of several methods for the 10-bar plane truss (g14). Our approach is labeled as CDE

Method	CONMIN	OPTDYN	LINRM	GRP-UI	SUMT
Weight	34800.0	N/A	33315.0	N/A	27564.0
Method	M-3	M-4	M-5	CDE	
Weight	26600.0	26654.0	26262.0	**20319.58**	

Table 5.20: Results of several methods for the 200-bar plane truss (g15). Our approach is labeled as CDE

Recursive Quadratic Programming (LINRM), Gradient Projection (GRP-UI), Exterior Penalty Function (SUMT), and Multiplier Methods (M-3, M-4 and M-5).

The parameters adopted by our cultured differential evolution approach are the same used to solve the benchmark. Thus, we performed the same number of objective function evaluations as before (100,100). For the 10-bar truss, the best result of the 30 independent runs is shown in Table 5.19, together with the results of the methods included in [5] (all the results presented are feasible). For the 200-bar truss, the best result of the 30 independent runs is shown in Table 5.20, with the results of other methods reported by Belegundu [5].

The best results of the cultured differential evolution are very competitive. Additionally, such results present a low variability over the 30 runs performed, as can be appreciated from Table 5.21.

Finally, we noticed that in engineering optimization problems with large search spaces (such as the two presented in this work), our approach could improve on the quality of the solutions produced if we allowed a larger number of fitness function evaluations. Although such increase may be unaffordable in real-world applications, this is a good indicative of the effectiveness of our approach, since, if possible, it keeps improv-

TF	Best	Mean	Worst	Std Dev
10-bar truss (g14)	4656.39	4656.52	4656.71	0.18
200-bar truss (g15)	20319.58	25393.37	30269.49	2492.24

Table 5.21: Statistics of the results obtained by our approach for the design of trusses

ing the solutions produced over time. To briefly illustrate this issue, we allowed our approach to run for 3500 generations (for a total of 350,100 fitness function evaluations) in the 200-bar truss problem. This setup produced a solution with a weight of only $\mathbf{15824.32}$, which is about 28% better than the solution reported in Table 5.20. Note however, that the computational effort required to produce this solution (measured in terms of fitness function evaluations) is over three times the original one.

6

Evolutionary Multiobjective Techniques

In the following chapters, we present an application of the previously developed cultural algorithm to solve multiobjective problems. In this chapter, we present a brief review of the previous related work in evolutionary multiobjective algorithms that is relevant to place our proposal in an appropriate context.

6.1 The multiobjective optimization problem

The statement of the multiobjective optimization problem is similar to the one of optimizing problems with only one objective. The problem consists of finding the decision vector $\mathbf{x}$ that optimizes:

$$\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x}))$$

where $\mathbf{f}$ is the vector of objective functions. The problem may or may not have constraints.

However, the concept of optimizing several functions simultaneously is not as simple as to find the optimum for each function considered separately, because the functions usually are in conflict with each other. Finding the optimum, then, can be interpreted as finding a good trade-off between all the objectives of the problem.

Thus, the notion of optimum in a multiobjective problem is ambiguous, since different people could claim that different points represent a good

trade-off, but this does not necessarily imply that such points are optimal. Vilfredo Pareto [130] introduced, towards the end of the 19th century, a more formal definition of optimality in multiobjective problems, which is known now as Pareto optimality. This definition is based on the concept of dominance, which is the following: a point $\mathbf{x}$ dominates another point $\mathbf{y}$ if and only if

$$f_i(\mathbf{x}) \leq f_i(\mathbf{y})$$

for $i = 1, 2, \ldots, m$, and, for at least one i:

$$f_i(\mathbf{x}) < f_i(\mathbf{y})$$

A point $\mathbf{x}^*$ is Pareto optimal, if there does not exist any other point $\mathbf{x}$ that dominates it. That is to say, $\mathbf{x}^*$ is Pareto optimal if there does not exist any other point $\mathbf{x}$ that fulfills:

$$f_i(\mathbf{x}) \leq f_i(\mathbf{x}^*)$$

for $i = 1, 2, \ldots, m$, and, for at least one i:

$$f_i(\mathbf{x}) < f_i(\mathbf{x}^*)$$

The previous definitions hold if the problem consists of minimizing all the objectives.

From the above definition we can see that a given problem not necessarily has a single Pareto optimal point. If the objectives are in conflict (as is often the case in practice), the problem has a set of points that fulfill this definition. After an optimization process has found several Pareto optimal points, the decision maker is the person who must choose the final solution that is to be implemented.

In addition to the definition of Pareto optimality, it is very common to find in specialized literature the term Pareto front. The Pareto front is the image of the set of points that conform the Pareto optimal set. In problems with continuous decision variables, the Pareto front usually is a partially continuous line (the set is infinite). Graphs of the Pareto front have been often used to show this characteristic, and also to show (in a graphical form) the trade-offs and the best results for each objective.

6.2 Evolutionary algorithms for multiobjective problems

Next, some of the most representative multiobjective evolutionary techniques developed to date are reviewed. The taxonomy of approaches adopted follows to the classification of Cohon and Marks [29], which has been widely accepted in the operations research community, and has been recently used to classify multiobjective evolutionary algorithms [27].

6.3 *A priori* techniques

The first group in this classification of techniques are the *a priori* techniques. This name is due to the fact that the preferences on the objective functions are given before the technique begins the search. There exist several a priori proposals, and some of them are reviewed next.

6.3.1 *Lexicographic ordering*

In this approach the objective functions are sorted according to their importance, and they are optimized in sequence beginning with the most important, and finishing with the less important. This technique is only advisable when the order of importance of the objectives is perfectly known, because the performance of the algorithm is highly dependent on this ordering.

Fourman [53], and Kursawe [91], for example, have proposed algorithms based on lexicographic ordering.

6.3.2 *Linear aggregating functions*

An aggregating function is the simplest way to adapt a single-objective optimization method for multiobjective problems, because it combines the results of the different objective functions into a single fitness value. This single value is often obtained by means of a linear combination of the objective functions, although that is not the only way to do it. This linear combination is done assigning a relative importance for each of the objective functions, through of a vector of constants with entries $w_i > 0$, where

all the weights adds to 1. The fitness function is:

$$fitness(\mathbf{x}) = \sum_{i=1}^{m} w_i f_i(\mathbf{x})$$

In most cases, it is necessary to introduce also a scaling factor, due to the incommensurability of the objective functions.

Some applications that have been reported using a linear aggregating function are reported on [168, 76, 106, 177].

Implementing an algorithm that uses a linear combination of objectives can be very simple, and also efficient, but a linear function is incapable to generate nonconvex portions of the Pareto front [35]. Moreover, it is usually difficult to perform a good assignment of weights for the objective functions.

6.3.3 *Nonlinear aggregating functions*

Although there exist several proposals and applications of nonlinear aggregating functions (for example, multiplicative functions), these have not been very popular. Perhaps the most used techniques have been those that use a vectors of goals (e.g., min-max [81, 161, 127, 170, 62, 28] and goal programming [13, 74, 174, 155]).

In the techniques with vectors of goals, the objective is to minimize the difference between the points generated by the search process, and a vector of desired objectives values. These techniques can generate, under certain circumstances, nonconvex Pareto fronts, in addition to being quite efficient computationally speaking. Nevertheless, if the characteristics of the problem are not known, it can be difficult to provide a good vector of goals.

6.4 Progressive techniques

In this sort of techniques, the preferences are expressed during the process of generating solutions. The decision maker will express how good is a solution for him/her, and the process must update the preferences to reflect the opinions of the decision maker, before the search process continues.

Although several techniques of this type exist in the operations research area, in evolutionary computation, progressive approaches are practically non-existing, although several *a priori* or *a posteriori* techniques have been modified to include the possibility of incorporating preferences during the search.

6.5 *A posteriori* techniques

In this set of techniques, the preferences are expressed at the end of the search process, and practically do not interfere with it. Here, an algorithm tries to generate solutions which represent all the trade-offs of the objective functions (that is to say, such techniques try to generate the entire Pareto front, or a uniform sample of it), and the decision maker will express the preferences when choosing the final solution.

6.5.1 *Criterion selection techniques*

In this type of techniques, the selection mechanism divides the population, and in each sub-population the selection is performed based on only one of the objectives. The best existing example of these techniques is Schaffer's vector evaluated genetic algorithm (VEGA) [156].

In VEGA, the total population is divided in m equal parts, and in each one of them the selection operates taking into account only one objective function. Once the selection mechanism was performed, the population is mixed to apply the rest of the evolutionary operators. All this process is repeated at each generation.

An evident VEGA problem is that it does not promote good trade-offs, but it prefers the best solutions of each objective separately. This problem is known as speciation (by its analogue in genetics). This problem was identified and attacked by Schaffer, using mating restrictions, not allowing recombination between individuals of the same sub-population, as well as other heuristic rules applied during the selection mechanism.

In another work [149] is also demonstrated that, if proportional selection is used, VEGA's scheme is equivalent to a linear combination of objective functions, which means that it has limitations regarding nonconvex Pareto fronts.

6.5.2 *Aggregating selection techniques*

Here is again the use of aggregating functions to solve multiobjective problems. The difference this time, is that the weights are varied through the generations or from an evaluation to another one, throughout the execution of the technique [75, 107]. The variation of the weights can be carried out in different ways, even encoding them in the chromosomes of the individuals.

Nevertheless, these techniques have the drawback that, if a linear combination is used, it is impossible to generate the entire Pareto front in all cases.

6.5.3 *Pareto sampling techniques*

This group of techniques works by identifying the nondominated individuals, and fitness assignment is made according to that characteristic.

There exist several forms to carry out the selection using Pareto sampling, but the original idea was developed by Goldberg [57]. He proposed to perform a ranking in the following way: first the nondominated individuals of the population are identified, and they are assigned the highest ranking value; they are ignored in the rest of the ranking process. The process continues identifying nondominated points in the remainder of the population, to which the next rank level is assigned. This process of ranking by layers is applied until a rank value has been assigned to each individual in the population.

Goldberg also suggested the use of niching and fitness sharing [57], to avoid convergence towards a single point. Since Goldberg's suggestion , the use of techniques to maintain diversity has become a general characteristic of multiobjective evolutionary algorithms, but also implies an additional computational cost.

The multi-objective genetic algorithm (*MOGA*) by Fonseca and Fleming [52], adopted a different ranking strategy. In this proposal, the rank of an individual depends on the number of individuals in the present population that dominate it, according to the following expression:

$$rank(\mathbf{x}_i, t) = 1 + p_i^{(t)}$$

where $p_i^{(t)}$ is the number of individuals that dominate $\mathbf{x}_i$ at generation t. From the previous expression is evident that the nondominated individu-

als will have a rank value of 1. After the ranking process, fitnesses are assigned through an interpolation of the ranking values. Fonseca and Fleming also use niching to avoid premature convergence. Fitness sharing is done in objective function space in this case.

Srinivas and Deb [162] proposed the nondominated sorting genetic algorithm (*NSGA*) that implements almost the exact ranking idea proposed by Goldberg. The algorithm proceeds with the ranking by layers, that they call *waves*. Also the fitness is shared (in decision variable space) between individuals close from each other in order to maintain diversity. A second version of this algorithm, the NSGA-II, was proposed by Deb et al. [41]. The NSGA-II incorporates elitism, and an operator to maintain diversity that does not require any parameters; it is computationally more efficient than the NSGA and has exhibited a very good performance, although some researchers have indicated that it performs considerably better with real encoded variables than with binary encoding [27].

Unlike the aforementioned algorithms, which rank the entire population, Horn and Nafpliotis [68] proposed a mechanism to perform tournaments based on Pareto dominance, with the aim of avoiding the computational cost of ranking all the population. Their proposal is called Niched-Pareto Genetic Algorithm (*NPGA*). In the NPGA, when two individuals compete, both are compared with a fraction of the population (the authors use 10%) and, if only one is nondominated, it wins; but if both of them are dominated or nondominated, then a niche count is performed, and the individual in a less populated region wins.

Later on, Erickson et al. [45] developed a second version of this algorithm, the NPGA 2, in which all the population is ranked before the tournaments. Another important difference is that the niche count is performed with the existing (partially filled) population of the next generation, whilst the original NPGA performs it with the current (complete) population. This way to perform the niche count with the next generation was previously proposed by Oei et al. [124].

The Strength Pareto Evolutionary Algorithm (*SPEA*), by Zitzler and Thiele [180], implements an external archive to store the nondominated solutions obtained throughout the evolutionary process. Each individual in this nondominated external archive is assigned a value of strength, which is proportional to the number of individuals it dominates. Then, the fitness of each individual in the population depends on the value of strength of all the individuals in the external archive that dominate it, and a series

of binary tournaments take place.

Also, a second version of SPEA (appropriately called SPEA2) was developed by Zitzler et al. [179]. In this algorithm, both the number of points that dominate an individual, and the number of points dominated by it, are used to calculate its fitness. Moreover, the method adopted to truncate the external archive in SPEA2 always preserves the ends of the Pareto front.

Van Veldhuizen and Lamont [172] proposed a multiobjective version of the messy genetic algorithm [37], called Multi-Objective Messy Genetic Algorithm (*MOMGA*). In the first phase, called initialization, the building blocks are generated exhaustively up to certain specified size. The next phase, called primordial, performs tournament selection. Finally, the juxtapositional phase generates the population by means of a cut and splice operator of recombination.

MOMGA-II was proposed by Zydallis et al. [181], and is the multiobjective version of the fast-messy genetic algorithm [58]. The initialization phase also produces building blocks, but this time are generated in a stochastic way, to avoid an uncontrolled growth of the population when generating all the existing building blocks. The second phase, called building block filtering, consists of reducing the population by means of a filtering operation, so that only the best building blocks are preserved; also, a tournament selection is performed in this phase. The third phase is the same found in the previous version.

Pareto-Based Selection

Among the approaches based on Pareto sampling are the techniques that use a selection based only on Pareto ranking, but they replace the other standard mechanisms, such as fitness sharing to maintain diversity. For example, the Thermodynamical Genetic Algorithm (*TDGA*) proposed by Kita et al. [86]. This algorithm is based on an idea similar to simulated annealing, and to perform selection it tries to minimize the free energy of the system; this helps to maintain diversity in the population. This method includes a temperature that is controlled according to a cooling schedule.

Another example of Pareto-based selection is the technique proposed by Osyczka and Kundu [128], called "distance method" because is based on the contact theorem to calculate relative distances among the individuals in the Pareto front. It does not require an additional method to main-

tain diversity.

Pareto-Demes Based Selection

The techniques within this group divide their population into several subpopulations, and they perform Pareto ranking on each one of them. This process is expected to be more efficient than other approaches because of this local ranking. Nevertheless, in order to obtain global Pareto optimal solutions, a mechanism of communication among them is needed. These techniques are specially suitable for parallelization [152].

Pareto Elitist-Based Selection

In evolutionary single-objective optimization, is necessary to retain the best individual in order to assure convergence. In evolutionary multiobjective optimization the role of elitism is not clearly known, but the experience shows that it is important, and most modern approaches include it. Elitism in the multiobjective optimization framework is more complicated than in the single-objective case, because in the latter case the best individual can be identified clearly, but in the former case all the nondominated individuals are equally good. The most common way to perform elitism in multiobjective evolutionary optimization is through an external archive, that stores the nondominated individuals to preserve them.

Elitism can be the main source to conform a population, using a secondary source (e.g., a diversity archive) to fill it. If this is the case, we say that the technique applies Pareto elitist-based selection. The following are some examples of approaches within this group.

The Pareto Archived Evolution Strategy (*PAES*), proposed by Knowles and Corne [87], is a $(1 + 1)$-ES with an external archive incorporated, in which the nondominated individuals found along the evolutionary process are stored. The nondominated individuals with respect to the current population are compared with the points in the external archive, and if they are again nondominated, they are stored. The mechanism adopted by PAES to maintain diversity consists of an adaptive grid, that is computationally more efficient than the niching methods.

Knowles and Corne also experimented with a $(1 + \lambda)$-ES and a $(\mu + \lambda)$-ES, but they claim that there are no significant improvements, but an increment in the computational cost associated [87].

Corne et al. [31] proposed the Pareto Envelope-based Selection Algorithm (*PESA*), with a small main population, and a larger secondary population (similar to the external archive mentioned before). During each iteration of the algorithm, some points from the external archive are randomly selected, and they are transformed to produce the new individuals in the main population; when this main population is filled, the nondominated individuals will be integrated to the secondary archive.

Later on, the PESA-II was proposed by Corne et al. [30], whose main difference with PESA is the region-based selection (the selection is performed using regions, and not individuals). When the mechanism selects a region as a parent, an individual within that region is selected at random.

The micro-Genetic Algorithm (μ-*GA*), proposed by Coello Coello and Toscano Pulido [24, 25], is another algorithm that keeps a small main population. It has a population memory, that is divided in a replaceable and a non-replaceable part; from this memory are selected the individuals to conform the main population. This small population is adopted by the genetic algorithm with normal operators, and when it converges, it provides the nondominated individuals to fill up an external memory (external archive). Some of the individuals in this external memory will enter to the replaceable part of the population memory. The μ-GA uses three types of elitism.

6.5.4 *Independent sampling techniques*

Independent sampling consists of performing several executions of the technique to find different points of the Pareto front.

The independent sampling techniques that have been proposed in evolutionary computation consist of scalarizing functions, but in this case the weights do not represent the preferences, but that they are varied to perform independent executions, so that different portions of the Pareto front can be found [117, 12, 171, 77, 132].

These techniques can generate good fronts, but they are not very popular probably because the computational cost may be excessive when many points are required, or when the problem has many objectives.

The approach proposed in this work can be classified within the independent sampling techniques. Our motivation is to take advantage of the convergence properties of the CDE approach, while managing the aforementioned drawbacks with additional mechanisms to reduce their influ-

ence.

It is worth mentioning that our approach performs several optimization processes, one for each point produced, but they are not totally independent, as we will see in the following chapter.

7

A Proposal for Multiobjective Optimization using the Cultured Differential Evolution

Hybrid approaches of evolutionary techniques and mathematical programming methods for multiobjective optimization are not very popular, probably due to the fact that mathematical programming techniques are frequently scalarizing functions, and thus require several executions of a single-objective optimizer to obtain the Pareto set or a sample of it. Conversely, most evolutionary multiobjective approaches obtain a set of non-dominated solutions (*i.e.*, an approximation of the Pareto set) in a single run.

The experience of some researchers is that hybrid approaches have been found to be relatively expensive when solving "easy" multiobjective problems, because of the several single-objective optimizations that need to be performed in order to generate the Pareto front.

Nevertheless, and despite their disadvantages, we consider that a hybrid approach can be a very effective choice under certain conditions. In our experience, mathematical programming techniques tend to produce points of very high quality (*i.e.*, tend to produce Pareto optimal points, or very good approximations of them), even when the problem may appear to be very difficult for most elitist multiobjective evolutionary algorithms based on Pareto ranking. This is due to the fact that in this case the search focuses on a single point, instead of aiming to converge to a set of them, and therefore, a better exploitation may take place.

In this work, we use the ε-constraint method as the mathematical programming technique, and the CDE previously described as the evolutionary algorithm. The CDE, according to our previous results, allows us to reduce the total number of fitness function evaluations, while obtaining competitive results, due to the extracted and applied information during the search process. Even with this reduction of function evaluations, the total approach may be more expensive (in fitness function evaluations) than a current multiobjective evolutionary algorithm, when solving "easy" multiobjective problems. But the advantages are evident when trying to solve "hard" problems.

When we talk about "easy" or "hard" problems, we correlate this terms to the degree of difficulty for a state-of-the-art evolutionary multiobjective approach (such as the NSGA-II) to solve it, with a parameter setup as recommended by its authors, and a typical budget of fitness function evaluations (10,000–100,000). Recently, researchers have proposed problem sets, that contain very difficult problems (according to the previous meaning), because with them, most evolutionary algorithms cannot converge to the true Pareto front within 100,000 fitness function evaluations or even more [71, 126]. It is precisely in these "hard" problems where a hybrid approach may appear advantageous, mainly when we use an efficient evolutionary approach for the hybridization, such as the CDE.

Moreover, in order to reduce the computational cost of the hybrid approach, we propose to generate only a few points. We will also show that, by carefully choosing the values of the ε vector, it is possible to obtain a well distributed set of points, that constitute a reasonably good approximation of the true Pareto front. These points are then processed by another approach able to diversify them such that a larger area of the Pareto front can be covered. Obviously, this diversification approach, which acts as a local search algorithm, should be computationally affordable, so that the total cost of the proposed hybrid algorithm remains reasonably low. We propose a simple evolutionary algorithm for this step.

With this further processing of the obtained points, we also alleviate some of the problems related to the parametrization of the ε-constraint method when dealing with many-objective problems.

Next, we will describe the hybridization of the ε-constraint method with the CDE, and the approach developed for diversification of the points. But first, some considerations to properly assign the values of the ε vector are provided.

7.1 Estimating the nadir objective vector

To assign different values to the ε vector without any knowledge of the problem, is not a trivial task. Recently, Laumanns et al. [101] proposed a method to vary the ε values. It consist of executing an initial optimization without constraints, and then use the objective functions values of the result to set up the values for ε for the next optimization, and so on. If the Pareto front is discrete, this approach is particularly suitable, because it can find the entire Pareto optimal set, as proved in [101].

This approach may be too expensive when the Pareto front is continuous, as it is most likely to be when we are dealing with real-valued problems. This is because the ε values tend to produce points that are too close of each other, making very difficult to control the number of points desired as outcome, and requiring many single-objective optimizations. As our approach is designed to work with real-valued problems, we considered an alternative: to obtain an approximation of the dimensions of the Pareto front, and then divide it into a number of intervals depending of the number of solutions that we want as outcome. The ε_j must vary from the best to the worst value for the objective j, *i.e.* the search must move from the ideal to the nadir objective vector.

The estimation of the ideal objective vector involves individual optimizations of one objective at a time. On the other hand, the estimation of the nadir objective vector is a more difficult task [118]. Currently, there are no efficient and reliable methods to estimate the nadir point, for an arbitrary problem. Only for the two-objective case, there exists a simple method that can provide a good estimation, which is called the *payoff table*. This approach consists of performing individual optimizations of one objective at a time, similar to those required to the ideal vector; the result of every optimization is evaluated for all the objectives, an finally the worst values for each objective are recorded. Figure 7.1 illustrates this process. A first approach for solving two-objective problems using this method was published in [99].

The payoff table method may produce estimations with significant error, when the problem has more than two objectives, regardless of the optimization procedure adopted. As we also want to solve problems with three or more objectives, the alternative is to use metaheuristics for this task as well. In this work, we use a technique based on the approach in

Results for the optimizations:

f_1	f_2	f_3	f_4
23.5	62.63	23.75	76.44
39.68	**48.68**	*84.84*	*186.64*
65.46	64.68	**8.80**	43.84
84.16	*92.34*	13.39	**14.39**

Estimated ideal objective vector:

23.5	**48.68**	**8.80**	**14.39**

Estimated nadir objective vector:

84.16	*92.34*	*84.84*	*186.64*

Figure 7.1: Example of the payoff table method

[39], which is a modification of the crowding mechanism of the NSGA-II. The modification consists of emphasizing the generation of nondominated solutions near to the edges of the Pareto front, and not only the extreme values. This is done checking first for nondominance; then, the points of a given front are sorted, and assigned a crowding distance higher for each of the extreme valued, and lower for central values. This procedure is performed for each objective, and the final crowding value of each point is assigned as the maximum of all the computed values for that point.

In this work, we perform the estimation using a standard differential evolution approach, incorporating the new ranking rules before the selection procedure. A detailed description of the fitness assignment mechanism is the following:

1. After the generation of all the children in the current iteration, perform Pareto ranking.

2. Sort the points of the first front using the values for the first objective.

3. Assign a fitness of 0 to the first and the last individuals; assign a fitness of 1 to the individuals next to the first and the last points, and so on.

4. Sort the same front using the values for the second objective, and repeat step 3. If the previous assigned fitness value of an individual is larger than the current one, do not update (keep the larger one). Sort for the other objectives and repeat step 3.

5. Repeat steps 2, 3 and 4 for the rest of the fronts. When assigning fitness to the k-th front, the fitness values will be: $t_1 + \ldots + t_{k-1}$ for the first and last points, $t_1 + \ldots + t_{k-1} + 1$ for the individuals next to the first and the last, and so on. t_k is the number of points in the k-th front.

The rest of the algorithm adopted is a standard differential evolution. The decision of using a differential evolution algorithm is due to the speed required, because this is only the first step of the process, and the estimation obtained will be relaxed in the next steps.

In the following, let us assume that the procedure $nadir\_st(\mathbf{f}, g)$ performs the modified differential evolution previously described for g generations. It will return two arrays, $\mathbf{lb}$ and $\mathbf{ub}$ with the estimated ideal and nadir objective vectors (assuming minimization).

7.2 The ε-constraint based approach: εCCDE

The single-objective optimizer, on which our method is based, is the cultured differential evolution previously described. Let's now assume that it is available as the procedure $cde(f_l, \varepsilon, g)$, which performs the optimization process of the ε-constraint method during g generations and returns the best point found. The pseudo-code of the ε-constraint with CDE (εCCDE) is shown in Algorithm 8.

In Algorithm 8, the lower and upper bounds, $\mathbf{lb}$ and $\mathbf{ub}$, are increased by a tolerance $\mathbf{t}$; this is done since the results of the $nadir\_st$ procedure are only approximations, and it is possible to find a better point beyond them. We use $t_j = 0.1(ub_j - lb_j)$. The ε values are updated with a δ, which depends on the number of points in the Pareto front desired by the user (or decision maker). It is obtained as follows: $\delta_j = \frac{ub_j - lb_j}{p_j}$. This way, we aim that the final points are equally spaced in their projection over the f_2 to f_m axes. g is an input parameter of the algorithm, but it is very important, because together with p_j and the population size of the cde procedure, μ, define the total number of fitness function evaluations required by the approach. The number of fitness function evaluations is $\left(\prod_{j=2}^{m} p_j\right) \cdot g \cdot \mu$.

Algorithm 8 shows f_1 as the objective to be optimized, and f_2 to f_m as the constraints. However, one can interchange the roles of the objectives

Algorithm 8: ε-constraint with CDE

$P = \emptyset$
$(\mathbf{lb}, \mathbf{ub}) = nadir\_st(\mathbf{f}, g)$
$\mathbf{ub} = \mathbf{ub} + \mathbf{t}$, $\mathbf{lb} = \mathbf{lb} - \mathbf{t}$
$\varepsilon_{j=2,\ldots,m} = lb_j + \delta_j$
while $\varepsilon_m \leq ub_m$ **do**
  $\mathbf{x} = cde(f_1, \varepsilon, g)$
  if $\mathbf{x}$ is nondominated with respect to P **then**
    $P = P - \{\mathbf{y} \in P \mid \mathbf{x} \succ \mathbf{y}\}$
    $P = P \cup \{\mathbf{x}\}$
  end if
  $\varepsilon_2 = \varepsilon_2 + \delta_2$
  for $j = 2$ to $m - 1$ **do**
    if $\varepsilon_j > ub_m$ **then**
      $\varepsilon_j = lb_j + \delta_j$
      $\varepsilon_{j+1} = \varepsilon_{j+1} + \delta_{j+1}$
    end if
  end for
end while

if the problem looks harder to solve in the original setting. In the experiments shown in this thesis, the original setting was always preserved, and f_1 was always taken as the objective to optimize, to allow a fair comparison. But, as a suggestion for better results, if it is known which of the objective functions is the most difficult to optimize, such objective function should be chosen to be optimized, and the rest should be adopted as constraints.

In order to improve the performance of each optimization process, the algorithm shares a percentage of the population, in the initial population of the next process. This helps because the problems to be solved are very similar, and the only change is the upper bound of the objective functions which are treated as constraints. When all the population is shared, the loss of diversity leads to premature convergence. In practice, we found that a small percentage (around 10%) of the population to be shared is enough to improve convergence without losing diversity.

The εCCDE approach provides good results by itself (see Chapter 8), but it can result computationally expensive when many points of the Pareto front are required (*i.e.*, when p_j is large), because it needs an individual optimization process for each point. At this point, we propose to keep the p_j values low, and to use an alternative technique to spread out the few points obtained earlier, in order to cover a larger area of the Pareto front.

By low, we mean $p_j \leq 5$, depending on the number of objectives. When the number of objectives m is greater than, say, 6, one may use $p_j = 1$ for $m - 6$ objectives, and $p_j = 2$ to the rest, to avoid adding unnecessary computational cost to this phase, even when this may affect the spread over the Pareto front. This is an inherent problem of the technique, and is related to the fact that, when dealing with many-objective problems, the search for Pareto optimal solutions is harder, because more points become nondominated.

This approach, based only on the ε-constraint method, and without any dispersion technique, was published in [99].

7.3 An additional technique for dispersion

There exist some techniques specialized in spreading points from the Pareto front. Here, we developed a simple evolutionary algorithm, based on the assumption that we have as input a well distributed sample of the Pareto

front. This sample is obtained with the εCCDE approach.

The algorithm then, tries to fill the spaces between one point and another. This is made using crossover operators based on interpolation, and then we apply a mutation operator. The procedure is described in Algorithm 9.

Algorithm 9: Evolutionary algorithm for disperssion

Set the input points as the initial population
repeat
  Select one crossover operator
  Randomly select as many parents as needed for the crossover operator
  Apply crossover
  Apply mutation
  if the newly generated point is nondominated **then**
    Add the new point to the population
  end if
  if Size of the population $>$ desired number points **then**
    Remove one point in the most populated area
  end if
until the maximum number of evaluations has been reached

Two crossover operators were used, based on linear and quadratic interpolation, respectively. The linear crossover operator is especially useful when the Pareto optimal set is a convex set. For other cases, the quadratic crossover operator, as well as the mutation operator complement the job.

7.3.1 *Crossover operators*

The linear crossover operator to generate the j-th child is the following:

$$\mathbf{x}'_j = (1-\alpha)\mathbf{x}_{r1} + \alpha\mathbf{x}_{r2}$$

where the random integers $r1, r2 \in \{1, \ldots, \mu\}$ represent the parents, and α is a random number with mean = 0.5 and standard deviation = 0.5. This is to keep most of the time the newly generated individual between $\mathbf{x}_{r1}$ and $\mathbf{x}_{r2}$, but with a small chance to generate it outside this interval (to make an extrapolation).

The quadratic crossover operator is the following:

$$\begin{aligned} x'_{1,j} &= (1-\alpha_3)x_{1,r1} + \alpha_3 x_{1,r3} \\ x'_{i=2,\ldots,n,j} &= (1-\alpha_1)(1-\alpha_3)x_{i,r1} + \alpha_1(\alpha_2-1)x_{i,r2} + \alpha_3\alpha_2 x_{i,r3} \end{aligned}$$

where $\alpha_1 = \frac{x'_1 - x_1^{r1}}{x_1^{r2} - x_1^{r1}}$, $\alpha_2 = \frac{x'_1 - x_1^{r2}}{x_1^{r3} - x_1^{r2}}$. α_3 must be obtained as α for the linear crossover operator, allowing sometimes to compute an extrapolation. In this case there are three parents, and the child obtained will be a quadratic interpolation in their projection over the x_1-x_i axes (for $i = 1, \ldots, n$).

7.3.2 *Mutation operator*

Finally, a very small mutation is applied, mainly not to destroy the effect of the crossover operators, but to perform a local search. To avoid the need of self-adaptation in its magnitude, we use the values of the parents:

$$x''_{i=1,\ldots,n} = x'_i + Z_i\left(0, 0.1\left|x_i^{r1} - x_i^{r2}\right|\right)$$

where $Z_i(\mu, \sigma)$ is a realization of a random variable, with mean μ and standard deviation σ.

This evolutionary algorithm is designed to generate several intermediate points over the Pareto front, assuming that the input points are Pareto optimal. In addition to some Pareto points, it takes as input parameter the limit for the fitness function evaluations it will perform, e_{max}. This is an alternative for a second phase of the approach in order to save evaluations, but it is not the only one. In [100], we use an approach based on rough sets theory.

7.4 Parameters of the technique

The parameters needed for this approach to work are summarized in Table 7.1. We suggest some values for each parameter, according to our observations, empirically derived after numerous experiments.

Besides the parameters in Table 7.1, is necessary to set the parameters of the CDE procedure, as described in Section 4.5 on page 46, including all the parameters related to the belief space.

In the next chapter, we will describe the performed experiments, and we will present the comparison of the results obtained.

Table 7.1: Parameters for the εCCDE technique

Parameter	**Meaning**	**Suggested Value**
μ	Population size of the individual optimizations.	As the CDE (Chapter 4), the approach works well with small populations. A good choice is between 10 and 50.
g	Maximum number of generations per each individual optimization.	Between 50 and 100. Try reducing it if more speed is needed, or increasing it if more precise results are required.
p_j	Number of divisions for the ε-constraint method per objective. It also refers to the number of points on each dimension.	The total number of points obtained during the ε-constraint phase is $\prod_{j=2}^{m} p_j$. It is advisable to keep this number $\leq$ 32, or lower (near to 10) if possible.
s_{share}	Portion of the population shared between optimizations.	Use a small value, such as 10 %, to improve the individual optimizations keeping a low risk of diversity loss.
$e_{\max}$	Maximum number of fitness function evaluations to be performed by the evolutionary algorithm for dispersion.	More evaluations provide more points and they tend to be closer to the true Pareto front. Perform as many as resources allow.

8

Results for the Multiobjective Optimization Approach

In the previous chapter, we mention that the εCCDE approach can be applied alone, producing all points as a result of individual optimizations. This may result expensive, especially if the problem has many objectives. In this chapter, we perform some experiments with εCCDE alone, on problems with only two objectives, in order to explore its performance. Then, we perform more experiments of the εCCDE and the additional technique for dispersion, on problems of two and three objectives.

8.1 εCCDE alone

In order to validate the performance of the proposed approach, some test functions have been taken from the specialized literature. One may think that the several single-objective optimizations required may give rise to a prohibitively high computational cost, which is unnecessary considering that a modern multiobjective evolutionary algorithm may produce a similar approximation of the Pareto front at a much more affordable computational cost. There are problems, however, where this is not the case, and in which a modern evolutionary algorithm presents difficulties, or definitively cannot converge to the true Pareto front even if we do not restrict the number of evaluations performed. It is precisely in those cases for which we believe that our approach can be a viable alternative. The

εCCDE, as a hybrid approach, focuses on a region during each individual optimization, and this may result on a good approximation for that individual point, even tough the rest of the front is ignored in that step (with several individual optimizations it is possible to cover a significant region of the Pareto front).

8.1.1 *Test problems*

To make evident the possible advantages of our approach, we looked within the current benchmarks for multiobjective problems that are particularly difficult to solve for current multiobjective evolutionary approaches. Our search led us to the use of a recent benchmark proposed by Huband et al. [70]. This benchmark was constructed using a block-oriented approach, where each block introduces a desired feature to the problem. For example, there are blocks for making the problem non-separable, deceptive, multimodal, etc. The shape of the Pareto front is also controlled with blocks, and it is possible to design linear, concave, convex, mixed or disconnected fronts.

In this benchmark we found very hard problems (WFG1, WFG2 and WFG9). Each of them has 24 variables. WFG1 is strongly biased toward small values of the first 4 variables, WFG2 is non-separable and also has a disconnected Pareto front, and WFG9 is a deceptive problem.

All the problems in the benchmark were used for the experiments. The expressions for the problems are provided in Appendix B.

We decided to compare results with respect to the NSGA-II [41], since this is an approach representative of the state-of-the-art in the area.

8.1.2 *Experimental setup*

We ran both algorithms during 25,000 fitness function evaluations each (except for WFG1). With this number of evaluations we can fairly perform the comparison (the authors of the NSGA-II performed close to 25,000 evaluations in the orignial proposal). We aimed to obtain a set of 50 points as a result of each run, so we adapted the parameters according to that. For the ε-CCDE, the parameters adopted were: $p = 50$, $g = 48$, with 10% of the population shared between optimizations (this 10% is chosen at random). For the cde procedure we used $\mu = 10$, $F = 0.7$, $CR = 0.5$. The population size of the NSGA-II was set to 52, and the number of generations to

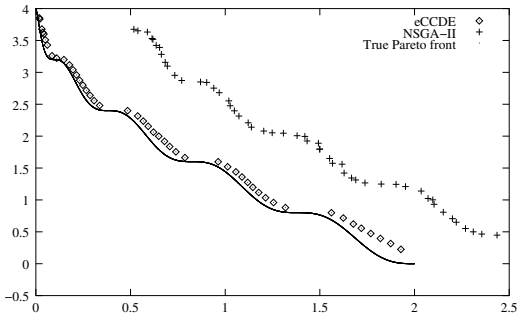


Figure 8.1: Results for the 2-objective WFG1

481. The rest of the parameters were set as recommended by its authors: probability of crossover $= 0.9$, probability of mutation $= 1/n$, the value of the distribution index for crossover $= 15$, and the value of the distribution index for mutation $= 20$.

Only for WFG1, the total number of fitness function evaluations was increased to 250,000, because this is a really difficult problem. The parameters adopted in this case were: $g = 120$ and $\mu = 40$. The number of generations of the NSGA-II was changed in this case to 4808. Even with this large number of iterations, the NSGA-II was not able to reach the true Pareto front.

In Figures 8.1 to 8.9, we show the results of a single run for each test problem. Since a visual comparison of the results may be inaccurate, we also used some performance measures to allow a quantitative comparison of results.

8.1.3 *Performance measures*

To assess the performance of the proposed approach, we adopted two measures.

Two Set Coverage (CS) measure [178], which is an indicator of how much a set covers (or dominates) another one. A value of $CS(X, Y) = 1$ means that all points in X dominate or are equal to Y. If $CS(X, Y) = 0$, there are no points in X that dominate some point in Y. When

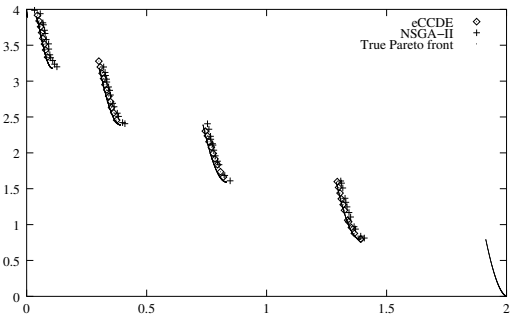


Figure 8.2: Results for the 2-objective WFG2

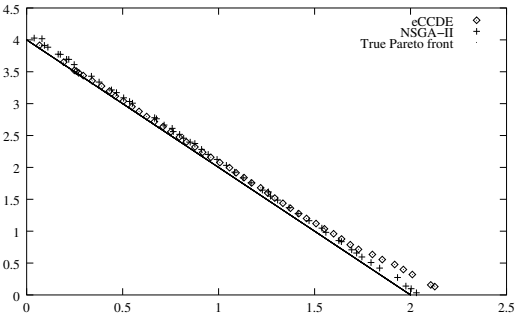


Figure 8.3: Results for the 2-objective WFG3

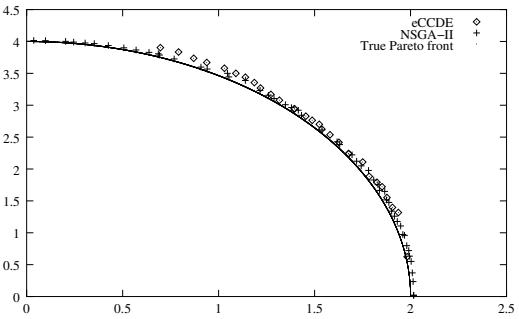


Figure 8.4: Results for the 2-objective WFG4

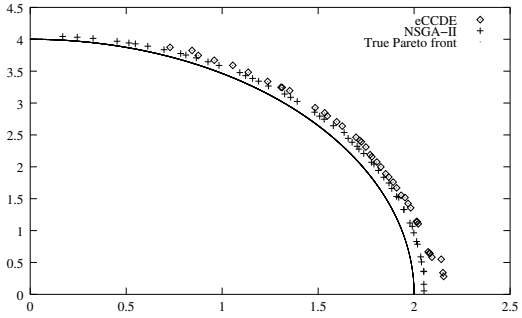


Figure 8.5: Results for the 2-objective WFG5

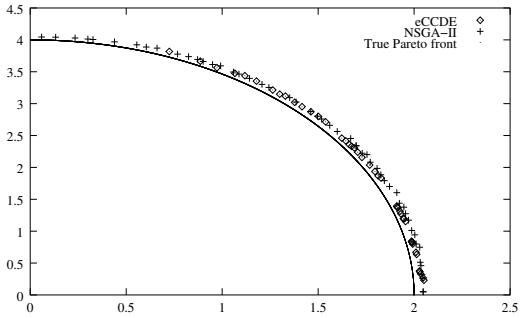


Figure 8.6: Results for the 2-objective WFG6

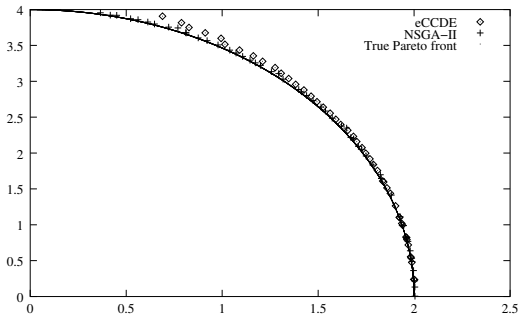


Figure 8.7: Results for the 2-objective WFG7

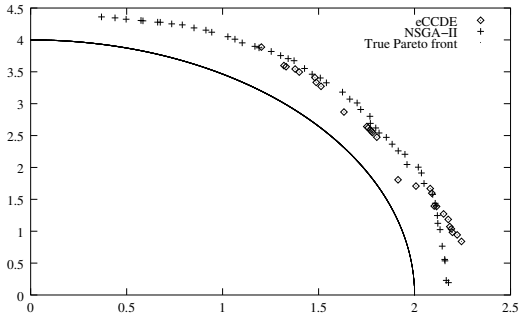


Figure 8.8: Results for the 2-objective WFG8

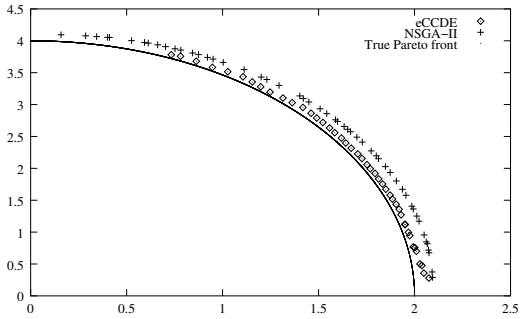


Figure 8.9: Results for the 2-objective WFG9

Table 8.1: Mean and standard deviation of the CS measure for the εCCDE alone (a larger value is better for the first algorithm)

Test Problem	$CS(\varepsilon\text{CCDE}, \text{NSGA-II})$ mean (std. dev.)	$CS(\text{NSGA-II}, \varepsilon\text{CCDE})$ mean (std. dev.)
WFG1	1.0000 (0.0000)	0.0000 (0.0000)
WFG2	0.8509 (0.1771)	0.0362 (0.0614)
WFG3	0.3987 (0.2691)	0.0908 (0.1368)
WFG4	0.2415 (0.1358)	0.4672 (0.2948)
WFG5	0.0065 (0.0073)	0.96694 (0.0461)
WFG6	0.3815 (0.1075)	0.04753 (0.0152)
WFG7	0.0124 (0.0062)	0.6275 (0.2546)
WFG8	0.6789 (0.1746)	0.3272 (0.2803)
WFG9	0.6415 (0.3669)	0.0995 (0.2114)

using this measure, it is necessary to calculate $CS(X, Y)$, as well as $CS(Y, X)$, to quantify possible overlaps between sets.

Binary Coverage (Q_c) [47], which is an indicator of the ability of an algorithm to obtain solutions near the extrema of the Pareto front, measuring the largest possible angle between two vectors of the output of an algorithm. This is a secondary criterion when proper convergence has been achieved, because it does not measure convergence itself, and it may occur that a set is very far from the true Pareto front, but it covers a larger area. A value of $Q_c(X, Y) > 0$ means that X contains points that are nearer to the extrema of the true Pareto front than those of Y (it covers a larger angle). It is worth noticing that $Q_c(X, Y) = -Q_c(Y, X)$, therefore, it is not necessary to compute both $Q_c(X, Y)$ and $Q_c(Y, X)$.

We executed our ε-CCDE 30 times per problem, and then executed the NSGA-II 30 times with the same random seeds, and we performed 30 one-to-one comparisons.

The results of the CS and the Q_c measures are summarized in Tables 8.1 and 8.2, respectively.

Table 8.2: Mean and standard deviation of the Q_c measure for the εCCDE alone (a larger value is better for the first algorithm)

Test Problem	$Q_c(\varepsilon\text{CCDE}, \text{NSGA-II})$ mean (std. dev.)
WFG1	0.2112 (0.0634)
WFG2	0.0677 (0.2172)
WFG3	-0.0299 (0.0253)
WFG4	-0.3967 (0.1469)
WFG5	-0.4375 (0.0168)
WFG6	-0.4328 (0.2258)
WFG7	-0.2813 (0.1868)
WFG8	-0.5678 (0.2687)
WFG9	-0.0913 (0.1154)

In more than half of the problems in Table 8.1, the εCCDE obtained better average values. However, the improvement is not always the same. In WFG1, all the points of εCCDE always dominate the points produced by the NSGA-II, because the latter cannot properly converge. The convergence is also good in problems WFG2 and WFG9. On the other hand, in WFG5 and WFG7, the NSGA-II presented a significantly better convergence. This may be due to the number of evaluations needed for the εCCDE to converge, even when the problem is not so hard.

In other cases, as WFG3, WFG4, WFG6 and WFG8, the algorithms are very competitive regarding convergence.

Now, we will examine Table 8.2. This time, our approach obtained the largest values for WFG1 and WFG2. For the rest of the problems, and particularly for WFG5, WFG6 and WFG8, this metric indicates that that NSGA-II can cover a larger length of the Pareto front. However, it is important to keep in mind that this is a secondary criterion, which becomes relevant only when a proper convergence has been achieved.

As an overall conclusion, we can see that when the problem is hard enough to prevent the convergence of an evolutionary multiobjective approach such as the NSGA-II, the εCCDE may achieve a better convergence and a better distribution. But, if the evolutionary algorithm can converge

within a moderate number of evaluations, the εCCDE alone may present difficulties for both, convergence and dispersion (within this number of evaluations).

8.2 εCCDE plus dispersion technique

This time, we perform a second phase of the approach, in order to reduce the number of fitness function evaluations necessary to obtain good results. The experiments were performed on two- and three-objective problems.

8.2.1 *Test problems*

We use again the WFG problems, but this time, we adopt the version with three objectives. The dispersion technique reduces the cost of the εCCDE, and we can now tackle three-objective problems at an affordable cost.

In addition to the WFG suite, in the recent literature there are other examples of hard problems for current multiobjective approaches, such as the problems from Okabe et al. [125]. This test suite consist only of two problems, but they have shown to be very challenging for current multiobjective evolutionary algorithms. These problems are constructed based on a starting space, and then applying transformations to obtain both, the Pareto front and the Pareto optimal set (*i.e.*, the optimal points in objective and decision variable space). The aim of this benchmark is to obtain problems with a Pareto optimal set whose expression is nonlinear in objective space, but also in decision variable space.

The problems of this benchmark are designated as OKA1 and OKA2. They only have 2 and 3 variables and 2 objectives, but the geometry of their Pareto optimal sets is nonlinear, and they are also strongly biased to the opposite side of the Pareto front.

We also use four of the so-called Zitzler-Deb-Thiele (ZDT) problems [178], because they have been used frequently in the specialized literature, and they constitute a reference point for many researchers in the field.

The expressions for all the problems are found in Appendix B.

Again, we compare our results with respect to the NSGA-II [41]. It is also known, that the NSGA-II performs particularly well in the ZDT problem set.

8.2.2 *Experimental setup*

We ran both algorithms during 15,000 fitness function evaluations each (including those evaluations required to estimate the ideal vector), except for OKA2 and WFG1. We aimed to obtain a set of 100 points as a result of each run, so we adapted the parameters according to that. For the εCCDE, the parameters adopted were, for the two-objective problems: $p = 5$, $g = 100$, with 10% of the population shared between optimizations (this 10% is chosen at random), for the cde procedure we used $\mu = 20$, $F = 0.7$, $CR = 0.5$.

The parameters for the three-objective problems (the parameters must be different because the number of evaluations depends of the number of objectives m) were: $p = 3$, $g = 70$ and $\mu = 16$.

The maximum number of evaluations performed by the dispersion technique was set to 5,000 in both cases.

The population size of the NSGA-II was set to 100, and the number of generations to 150. The rest of the parameters were set as recommended by the NSGA-II's authors: probability of crossover $= 0.9$, probability of mutation $= 1/n$, the value of the distribution index for crossover $= 15$, and the value of the distribution index for mutation $= 20$.

Only for OKA2 and WFG1, the total number of fitness function evaluations was increased to 25,000, because these are really difficult problems. In this case, we adopted, for the two-objective problem (OKA2): $g = 150$, and for the three-objective problem (WFG1): $g = 104$. In both cases the maximum number of evaluations of the dispersion technique was set to 10,000. The number of generations of the NSGA-II was changed in this case to 250, to allow a fair comparison. However, even with this large number of iterations, the NSGA-II was not able to reach the true Pareto front of WFG1, and in the case of OKA2, a very small length of the Pareto front was covered.

In Figures 8.10 to 8.24, we show the results of the run on the median with respect to the CS measure for each test problem adopted.

8.2.3 *Performance measures*

The methodology to measure the performance of the algorithms was the same as the adopted for the εCCDE alone. The results are summarized in Table 8.3 for the CS measure, and in Table 8.4 for the Q_c measure.

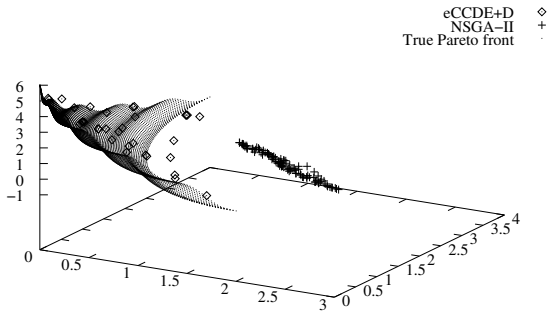


Figure 8.10: Results for the 3-objective WFG1

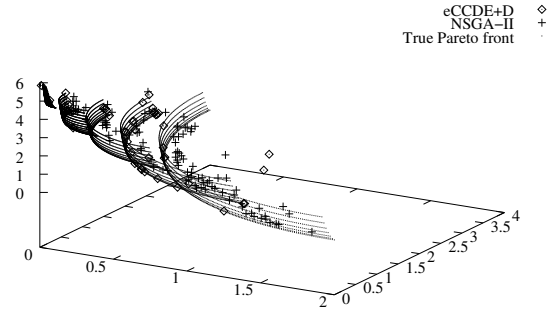


Figure 8.11: Results for the 3-objective WFG2

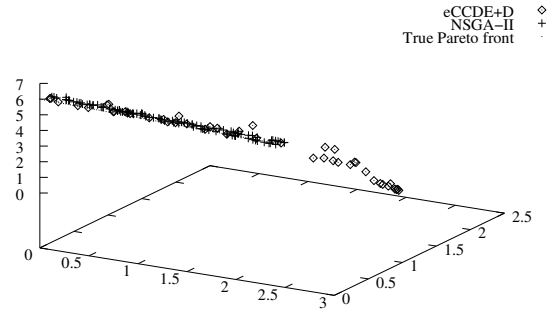


Figure 8.12: Results for the 3-objective WFG3

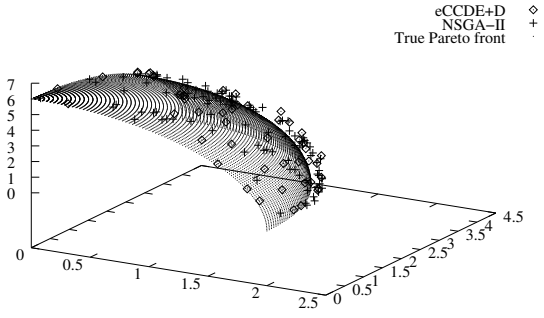


Figure 8.13: Results for the 3-objective WFG4

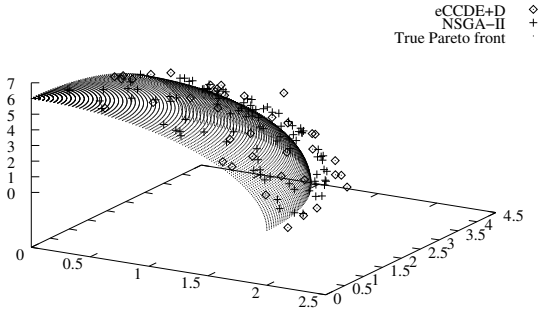


Figure 8.14: Results for the 3-objective WFG5

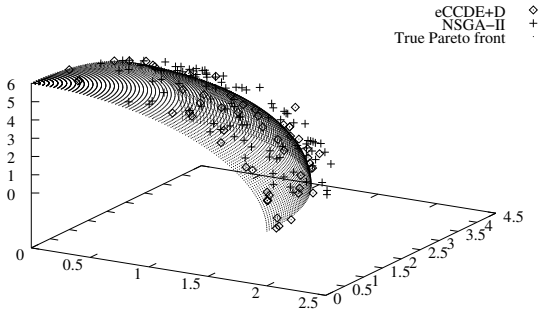


Figure 8.15: Results for the 3-objective WFG6

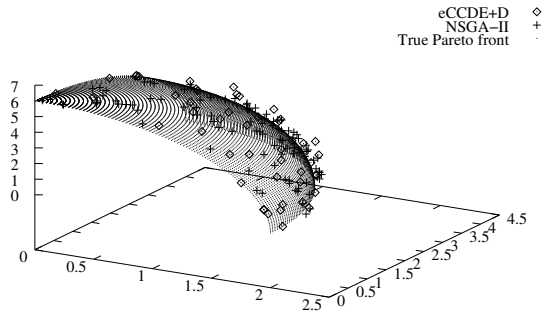


Figure 8.16: Results for the 3-objective WFG7

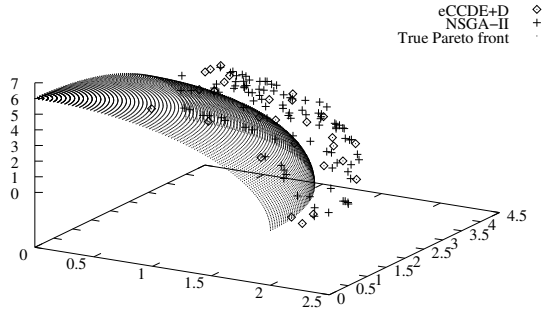


Figure 8.17: Results for the 3-objective WFG8

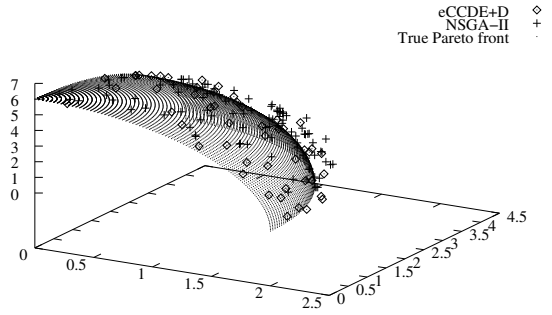


Figure 8.18: Results for the 3-objective WFG9

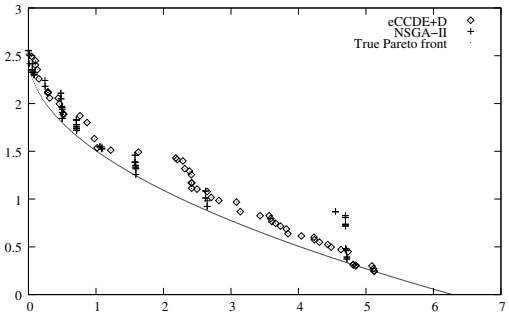


Figure 8.19: Results for OKA1

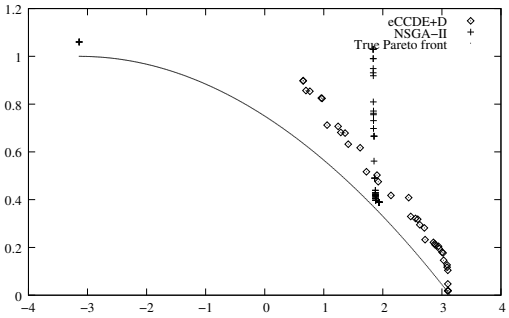


Figure 8.20: Results for OKA2

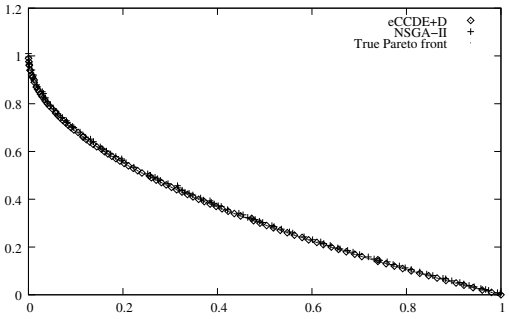


Figure 8.21: Results for ZDT1

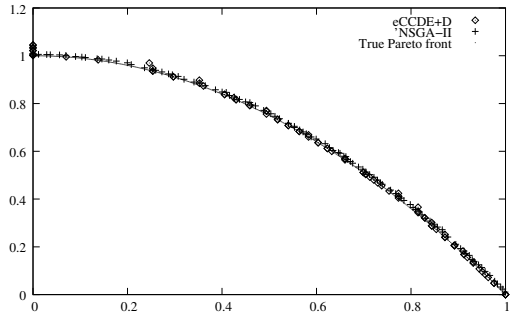


Figure 8.22: Results for ZDT2

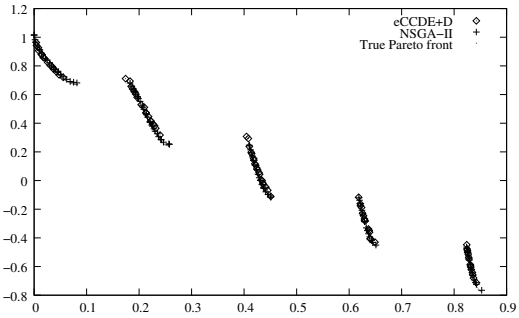


Figure 8.23: Results for ZDT3

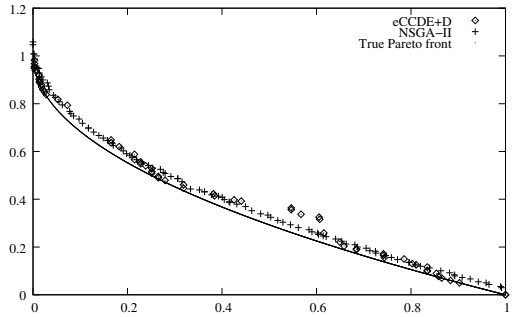


Figure 8.24: Results for ZDT4

Table 8.3: Mean and standard deviation of the CS measure for the εCCDE+D (a larger value is better for the first algorithm)

Test Problem	$CS(\varepsilon$CCDE+D, NSGA-II) mean (std. dev.)	CS(NSGA-II, εCCDE+D) mean (std. dev.)
WFG1	0.8040 (0.0169)	0.0000 (0.0000)
WFG2	0.5288 (0.4904)	0.3986 (0.3737)
WFG3	0.1273 (0.0387)	0.4146 (0.2638)
WFG4	0.1688 (0.0386)	0.1285 (0.0315)
WFG5	0.0033 (0.0027)	0.2723 (0.0783)
WFG6	0.2564 (0.1670)	0.0579 (0.3490)
WFG7	0.0021 (0.0015)	0.1683 (0.0872)
WFG8	0.2736 (0.1543)	0.1448 (0.0984)
WFG9	0.6896 (0.2014)	0.0543 (0.0539)
OKA1	0.2798 (0.2179)	0.1600 (0.1384)
OKA2	0.4165 (0.1593)	0.1688 (0.1062)
ZDT1	0.4896 (0.0738)	0.1906 (0.0760)
ZDT2	0.0269 (0.0094)	0.4000 (0.1724)
ZDT3	0.0242 (0.0083)	0.5145 (0.0574)
ZDT4	0.5906 (0.2208)	0.1966 (0.1118)

Table 8.4: Mean and standard deviation of the Q_c measure for the εCCDE+D (a larger value is better for the first algorithm)

Test Problem	$Q_c(\varepsilon\text{CCDE+D}, \text{NSGA-II})$ mean (std. dev.)
WFG1	0.6194 (0.1322)
WFG2	-0.1636 (0.1154)
WFG3	0.4158 (0.1667)
WFG4	-0.0711 (0.0241)
WFG5	-0.3513 (0.1784)
WFG6	-0.4257 (0.1686)
WFG7	-0.1687 (0.1268)
WFG8	0.1273 (0.0613)
WFG9	0.0361 (0.1028)
OKA1	0.0918 (0.1001)
OKA2	-0.1375 (0.1403)
ZDT1	0.0060 (0.0082)
ZDT2	0.0378 (0.0248)
ZDT3	-0.1102 (0.0615)
ZDT4	-0.4357 (0.3510)

In most of the problems in Table 8.3, the approach presented here obtained better average values. However, the improvement is not always the same. In WFG1 and WFG9, almost all the points of εCCDE always dominate the points produced by the NSGA-II, because the latter cannot properly converge. On the other hand, in WFG2, WFG4, and WFG8, as well as OKA1, both algorithms show difficulties to dominate the results of each other.

Regarding the ZDT problems, the NSGA-II presented a better convergence in two problems (ZDT2 and ZDT3), while in the other two our approach obtained a better convergence.

The results for the Q_c measure was this time more balanced. Our approach obtained the largest value for WFG1. For WFG5 and WFG6, as well as for ZDT4, this metric indicates that the NSGA-II can cover a significantly larger length of the Pareto front. But the previous measure for these problems, showed that our εCCDE obtained a better convergence than the NSGA-II, except in WFG5, where the NSGA-II obtained both, a better convergence and a larger length over the Pareto front.

Our approach presented good results on the two-objective version of WFG3, but in the three-objective version, the performance is not so good. This may be consequence of the degenerated Pareto front, a characteristic which is not evident on two objectives. Because of the division of the search space by the ε values, we expected the approach to have some problems with degenerated Pareto fronts.

Finally we can say that, with the addition of a dispersion technique, which reduces the number of function evaluations needed for the εCCDE to work, the approach becomes very competitive while preserving its advantages on hard problems.

9

Incorporation of Preferences to εCCDE

Incorporating preferences of the decision maker to the optimization process is a common practice in operational research [118]. It consist of expressing the preference for some objectives over the others, or for some region of the objective space, aiming to reduce the total number of solutions representing different compromises, and thus helping to the decision making process.

Evolutionary algorithms are frequently designed to obtain a sample of the whole Pareto front, giving all the information to the decision maker and leaving to him/her the decision of which to implement (*a posteriori* methods) [21]. However, it may be useful to express preferences *a priori* or interactively to the optimization process, for example to reduce the computational cost of generating many points on regions of little interest.

Here, we develop two mechanisms of incorporation of preferences to our previously proposed εCCDE. We obtain two main advantages from the use of such mechanisms:

- To reduce the computational cost related to obtaining the ideal and nadir objective vectors.

  The ideal and nadir objective vectors are not necessary when information about the region of desired solutions is available, because individual optimizations are carried out only inside that region.

- To alleviate the issues of the algorithm when solving many-objective

problems.

Because the number of desired solutions as outcome may be smaller (because they are closer to the desired values), is not necessary to perform an aggressive sub-division of the objective space. This point will be clarified after the descriptions of the developed mechanisms.

The two approaches developed for incorporation of preferences to the εCCDE are: to provide ranges for $m-1$ objectives, and to provide a vector of goals. Both approaches are described next.

9.1 Provide ranges for $m-1$ objectives

With this first approach is easier to control the length of the portion of the Pareto front that will be obtained, but the values of the solutions in one objective function will be defined by the other objectives. This is useful if the decision maker is interested in some values of specific objectives, but he/she has not information about all of them.

This approach is a direct application of the ε-constraint method, which needs the ideal and nadir vectors to bound the search. In this case, the search is bounded by the ranges provided by the decision maker. In Figure 9.1 is shown an example for two objectives. The values of f_1 are defined by the range given for f_2, which is the range of concern of the decision maker. A series of individual optimizations are carried out, as many as the number of points desired as outcome.

The implementation of this approach is very simple: the ranges provided by the decision maker are assigned to the **lb** and **ub** vectors in Algorithm 8 on page 86.

If many points are desired as outcome, or the problem has many (three or more) objectives, the ε-constraint method can be used only to find the extreme points in the range of interest (2^{m-1} points), and then the dispersion technique described in Section 7.3 on page 87 can be used.

9.1.1 *Results*

For this approach, it is very difficult to make a comparison of results with another multiobjective approach, because even when there are some approaches that incorporate preferences of the decision maker [33, 48, 144],

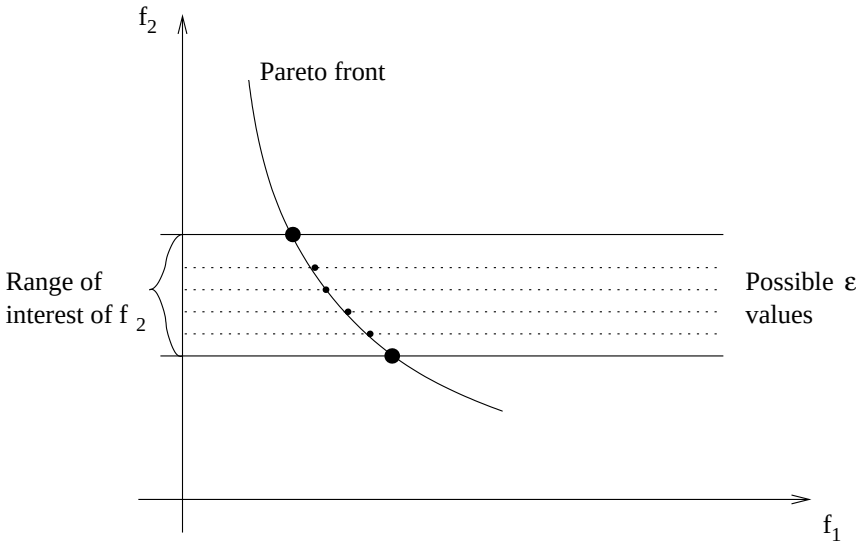


Figure 9.1: The ε-constraint method with ranges

the resulting covered region of the Pareto front will not be the same. Thus, we decided to measure only convergence, adopting a unary performance measure. We use the generational distance, GD [173], which measures the differences from the obtained points to the nearest points of the true Pareto front. As it measures distances, a value closer to zero is better.

The GD measure has received some critics because it only measures distance from the true Pareto front, and not the dispersion or region covered. The inverted GD measure somehow alleviates these issues, measuring the differences from every point of a good sample of the true Pareto front to the nearest obtained point. However, when only a portion of the Pareto front is intentionally produced, the inverted GD will always report bad results, whilst the original GD measure will only measure convergence, regardless of the portion of the Pareto front generated.

To validate this approach, we only use some WFG bi-objective problems, because the next approach is more suitable for many-objective problems. The ranges for the objective f_2 were obtained randomly between the corresponding entries of the ideal and nadir objective vectors. They are

Table 9.1: Ranges adopted for the experiments

Test problem	Ranges for the objective f_2
WFG1	0.99 – 1.78
WFG2	2.18 – 2.89
WFG2	0.68 – 2.38

Table 9.2: Mean and standard deviation of the GD measure for the incorporation of preferences through ranges (a smaller value is better)

Test problem	GD measure mean (std. dev.)
WFG1	0.0265 (0.0082)
WFG2	0.0030 (0.0011)
WFG2	0.0085 (0.0024)

shown in Table 9.1.

The rest of the parameters were adapted to obtain 10 points: $p = 10$, $g = 75$, with 10% of the population shared between optimizations (this 10% is chosen at random), for the cde procedure we used $\mu = 20$, $F = 0.7$, $CR = 0.5$. With these parameters, the approach performed $15,000$ function evaluations.

The results for WFG1, WFG2 and WFG3 are shown in Figures 9.2 to 9.4.

We executed the technique 30 times for each problem, with the same ranges, but different random seeds. The results of the application of the GD measure are in Table 9.2.

All the values of Table 9.2 are very small, indicating a good performance. The larger values correspond to WFG1; however, we had anticipated this result, because WFG1 has been shown to be a very difficult problem by other authors.

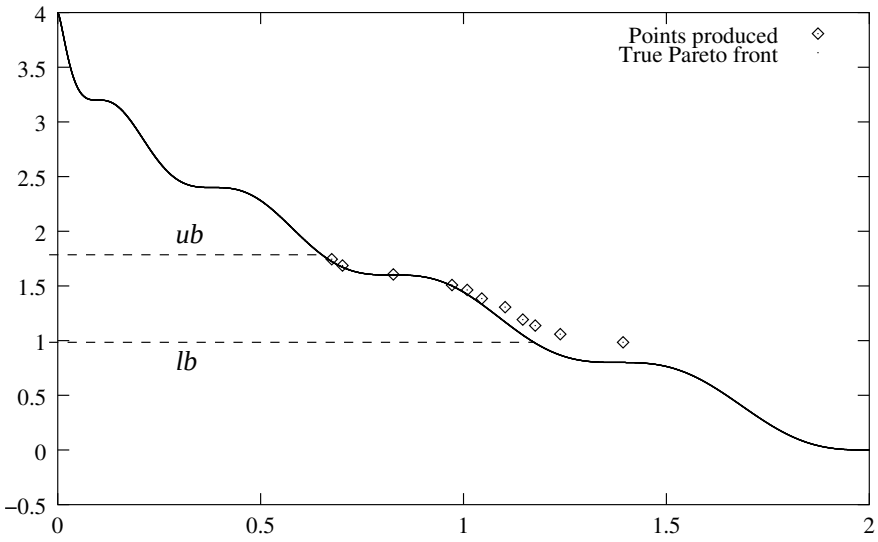


Figure 9.2: Results for WFG1

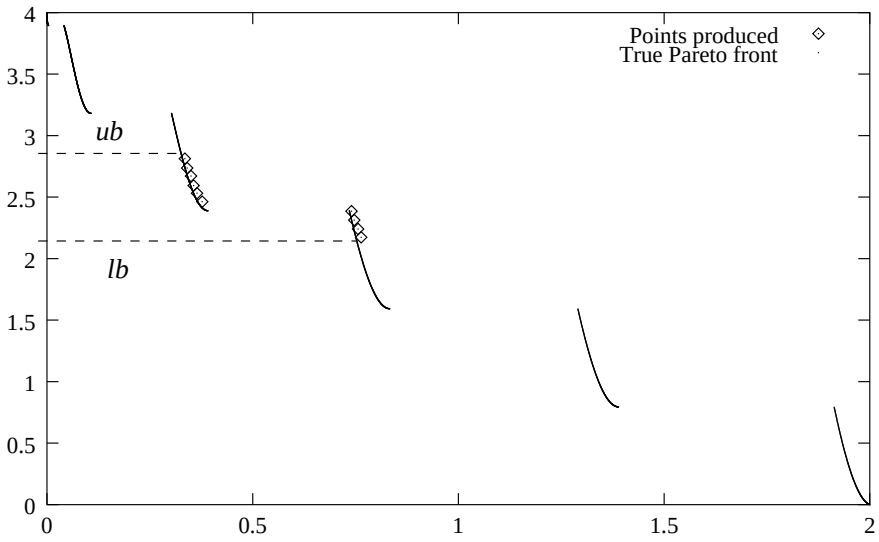


Figure 9.3: Results for WFG2

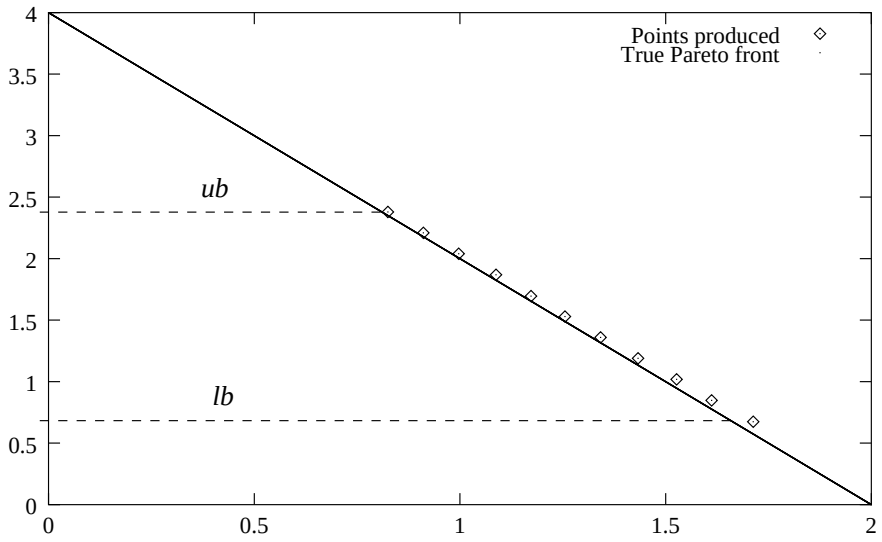


Figure 9.4: Results for WFG3

9.2 Provide a vector of goals

This second approach may appear more natural for a decision maker, because he/she provides only one point $\mathbf{z}^{\text{goal}} = (z_1^{\text{goal}}, \ldots, z_m^{\text{goal}})$ (one value for each of the m objectives), and expects values near of it. The entries of the $\mathbf{z}^{\text{goal}}$ vector are frequently called *aspiration levels* [118] or *goals*, and there are several methods based on them in the operational research area (such as goal programming, or the reference points models). For this reason, this approach may be more familiar to the decision maker.

With this approach is a little harder to control the length of the portion of the Pareto front that will be generated, but if at least two executions are allowed, or previous knowledge about the dimensions of the Pareto front is available, it is possible to determine such a length. However, this knowledge is not mandatory to apply this approach, and the resulting points will always be close to the provided goals.

Figures 9.5 and 9.6 show two cases of vectors of goals, one of them after the true Pareto front, and one before it. In both cases, an optimization per objective is performed, using a modified version of the ε-constraint problem, whose values for the constraints are taken from the vector of goals (for that reason, we call it *goal-constraint*):

$$\begin{aligned} &\text{minimize} && f_i(\mathbf{x}) \\ &\text{subject to} && f_j(\mathbf{x}) \leq z_j^{\text{goal}} \quad \forall j = \{1, \ldots, m\}, j \neq i \end{aligned}$$

This process is carried out for all $i \in \{1, \ldots, m\}$, with the aim of obtaining the vertices of the region to be explored. The solution of each goal-constraint problem is the nearest weakly Pareto optimal solution less or equal to the set of goals z_j^{goal} with $j \neq i$. If the solution of the problem is unique, it will be Pareto optimal, with the same proximity to the goals. These two characteristics can be proven in a very similar way as the proofs of Pareto optimality for the ε-constraint problem (see Appendix D).

It is possible that any of these optimization processes cannot find a feasible solution (depending on the shape of the Pareto front, and the vector of goals chosen), but an infeasible solution will work in most of the cases as an approximation for the next steps. If there is no feasible solution for one problem, it is possible to generate one by eliminating some of the $m-1$ constraints of the problem, but this may increase the computational cost of the approach. The only advise for this approach, is that the defined goals

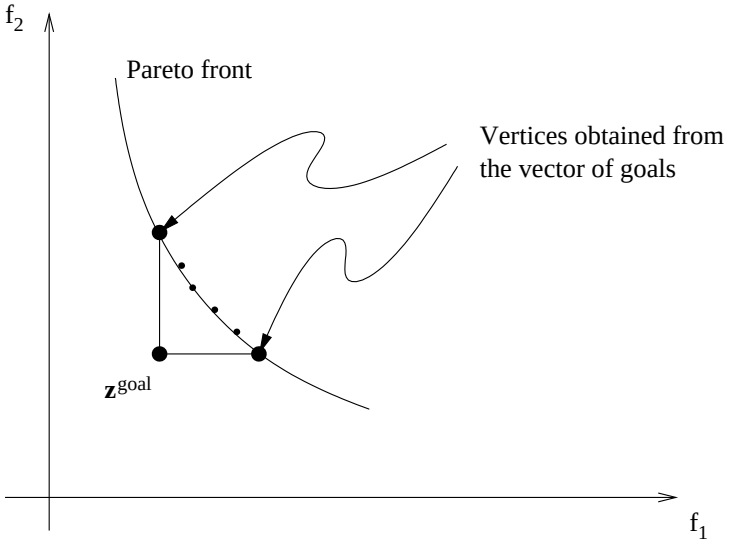


Figure 9.5: Use of a vector of goals after the Pareto front

are not lower that the corresponding entry of the ideal objective vector (in that case, no feasible solution will be found, because there is no x that fulfills the constraints of the problem).

Once all the optimizations have been performed, we apply the payoff table method to the results, and these values will define the ranges for the optimizations, together with the vector $\mathbf{z}^{\text{goal}}$.

We propose this goal-constraint method for defining the portion of the Pareto front generated by our approach, but it can be used by any other technique that supports ranges for the incorporation of preferences. The use of the same method for defining the generated portion of the Pareto front would allow to make comparisons of techniques, which is currently very difficult due to the variety of methods.

To obtain some points between these ranges (if desired by the user or decision maker), several executions of the ε-constraint method can be performed, or, again, the alternative dispersion technique can be used if the problem has many objectives. In such a case, we can only obtain the vertices of the region defined by $\mathbf{z}^{\text{goal}}$, and perform a few evaluations of the

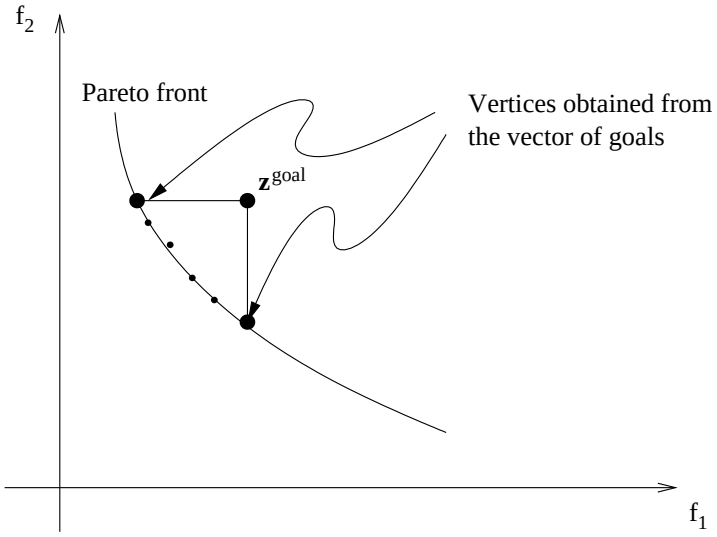


Figure 9.6: Use of a vector of goals before the Pareto front

dispersion technique, to obtain at least one intermediate point. This process will produce several points at a reasonable computational cost, even for problems of ten or more objectives.

9.2.1 *Results*

This time we performed the experiments on problems with more objectives, to show the potential of the technique. Firstly, we performed experiments on WFG1 and WFG2 with two and three objectives, to show the results graphically. Then, some experiments were performed on the same test problems with five and ten objectives.

The vectors of goals provided were obtained randomly between the ideal and nadir objective vectors. In Table 9.3 are shown the vectors of goals adopted. The rest of the parameters were adopted to obtain 20 points.

The parameters adopted for the two-objective problems were: $p = 5$, $g = 150$ for WFG1 and $g = 75$ for WFG2, with 10% of the population shared between optimizations (this 10% is chosen at random). For the cde

Table 9.3: Vectors of goals adopted for the experiments

Test Problem	Number of objectives	Vector of goals
WFG1	2	(0.47, 1.03)
WFG2	2	(1.08, 2.12)
WFG1	3	(0.23, 2.44, 4.91)
WFG2	3	(0.29, 0.79, 0.77)
WFG1	5	(1.48, 1.80, 1.90, 4.78, 7.03)
WFG2	5	(1.75, 0.58, 5.14, 0.91, 6.54)
WFG1	10	(1.97, 3.35, 0.40, 1.41, 8.35, 2.77, 2.80, 4.02, 14.67, 7.18)
WFG2	10	(1.50, 3.70, 5.90, 3.84, 9.84, 9.78, 1.33, 13.10, 17.44, 8.30)

procedure we used $\mu = 20$, $F = 0.7$, $CR = 0.5$. The maximum number of function evaluations for the dispersion technique was set to $7,500$. With these parameters, the approach performed $30,000$ function evaluations for WFG1 and $15,000$ for WFG2.

The parameters adopted for the three-objective problems were: $p = 3$, $g = 150$ for WFG1 and $g = 75$ for WFG2, with 10% of the population shared between optimizations (this 10% is chosen at random). For the cde procedure we used $\mu = 20$, $F = 0.7$, $CR = 0.5$. The maximum number of function evaluations for the dispersion technique was set to $6,500$. With these parameters, the approach performed $40,000$ function evaluations for WFG1 and $20,000$ for WFG2.

The parameters adopted for the five-objective problems were: $p = 2$, $g = 100$ for WFG1 and $g = 50$ for WFG2, with 10% of the population shared between optimizations (this 10% is chosen at random). For the cde procedure we used $\mu = 20$, $F = 0.7$, $CR = 0.5$. The maximum number of function evaluations for the dispersion technique was set to $4,000$. With these parameters, the approach performed $40,000$ function evaluations for WFG1 and $20,000$ for WFG2.

The parameters adopted for the ten-objective problems were: $p = 1$, $g = 150$ for WFG1 and $g = 75$ for WFG2, with 10% of the population shared between optimizations (this 10% is chosen at random). For the cde

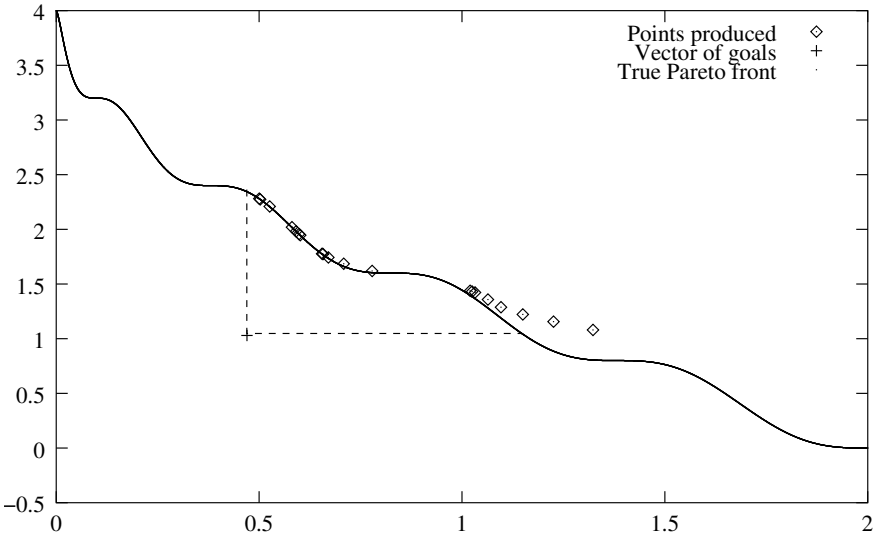


Figure 9.7: Results for the two-objective WFG1

procedure we used $\mu = 20$, $F = 0.7$, $CR = 0.5$. The maximum number of function evaluations for the dispersion technique was set to $10,000$. With these parameters, the approach performed $50,000$ function evaluations for WFG1 and $25,000$ for WFG2.

The results for the two-objective problems adopted are shown in Figures 9.7 and 9.8, while the results for the three-objective problems are shown in Figures 9.9 and 9.10.

The results of the application of the GD measure on all the instances adopted are in Table 9.4.

According to the GD measure, the approach decreases in quality as we increase the number of objectives. This sort of behavior is exhibited by most other evolutionary multiobjective approaches as well, and is subject of current research.

Another source for the degradation of the quality is that, as the number of objectives increases, the ranges for the higher objectives also increase (for the WFG problems), as can be seen from their expressions (see Appendix B).

Regarding the different test problems, the technique obtain better re-

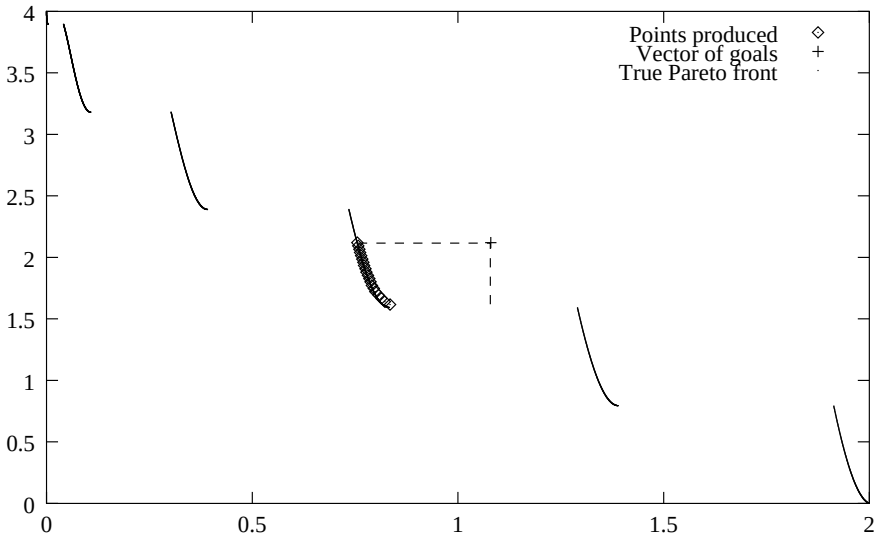


Figure 9.8: Results form the two-objective WFG2

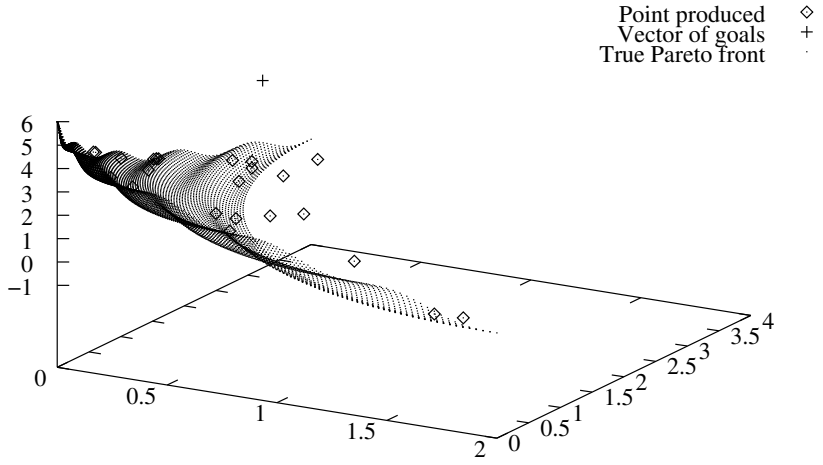


Figure 9.9: Results for the three-objective WFG1

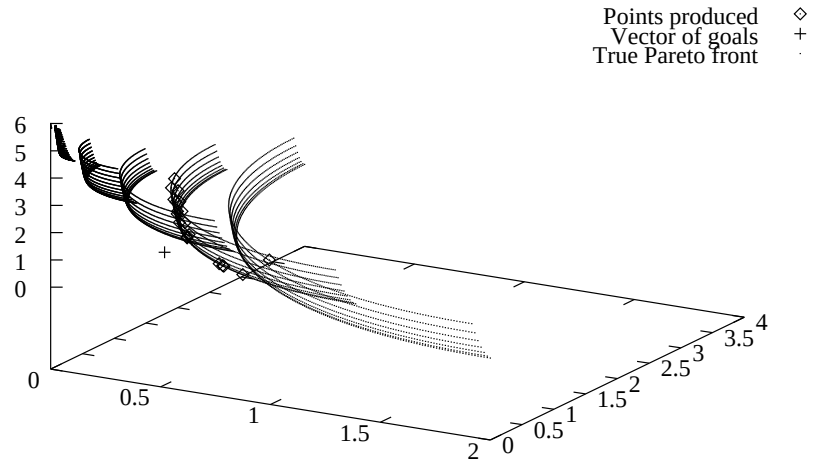


Figure 9.10: Results for the three-objective WFG2

Table 9.4: Mean and standard deviation of the GD measure for the incorporation of preferences through a vector of goals (a smaller value is better)

Test Problem	Number of objectives	GD measure mean (std. dev.)
WFG1	2	0.0142
WFG2	2	0.0015
WFG1	3	0.0511
WFG2	3	0.0124
WFG1	5	0.4176
WFG2	5	0.2379
WFG1	10	1.2867
WFG2	10	0.6675

sults for WFG2, for all the number of objectives adopted here. These results are consistent with all the previous experiments involving WFG1, which has shown to be a very difficult problem.

The overall results are very good, lower than 0.1 for three objectives, lower than 0.5 for five objectives, and lower than 1.5 for ten objectives. When comparing these values is important to keep in mind the increase of the ranges of higher objectives for WGF problems.

10

Final Remarks

In the first part of this work, we developed a cultural algorithm for constrained optimization problems. This algorithm is based on differential evolution, a recently developed algorithm that has been found to be very effective for problems with real-valued decision variables. The cultural algorithm proposed here, includes four knowledge sources to exploit different aspects of the search, cooperating and competing to be the one with more improvements on the solutions. The knowledge sources included are the following:

Situational knowledge. This source involves the identification of a leader, which is the best individual found so far, and the attempts of other individuals to imitate it. Situational knowledge increases the pressure of the search towards the best point found; this has found to be useful in certain phases of the search. Moreover, the variant of differential evolution which only has an operator similar to this one, is one of the most competitive variants.

Normative knowledge. This source stores ranges where good solutions have been found. It has two adaptive functionalities: at the beginning of the search, or when the ranges are wide, it works as an exploration operator, because it tries to spread the solutions into the intervals; if the ranges are narrow, it works as an exploitation operator, because it focus only on that good region. Our implementation

for differential evolution can also be seen as a mechanism for self-adaptation of the F parameter.

Topographical knowledge. It builds a map of promising regions, based on the fitness of previously generated points. Thus, it is possible to identify disjoint promising regions. This source also increases the pressure of the search, but this pressure is towards several regions (possibly different local optima), diversifying the search.

History knowledge. This source tries to identify patterns in the locations of previous good solutions, in order to predict unexplored good regions. The original motivation of this source is for its use in dynamic environments, but it could be useful also in a static problem which present certain regularity or symmetry in its fitness landscape. The proposed expression for this knowledge source provides also a mechanism to preserve diversity, generating random individuals in the whole search space.

The synergic effect of all knowledge sources has shown to be effective, when analyzing the results of the proposed approach on all the problems of a well-known benchmark; also the approach was tested on two different instances of an engineering optimization problem (design of trusses).

In the second part of the work, we explore further alternatives for the application of cultural algorithms. This time, we couple the previously developed cultural algorithm and a mathematical programming method (the ε-constraint method), in order to solve multiobjective optimization problems. This time the motivation is to take advantage of the convergence capabilities that the cultural algorithm exhibited, assigning it task of sampling a Pareto front. This way, we expect to obtain good approximations of the Pareto front, even in problems where other evolutionary approaches have presented difficulties.

Regarding computational cost, with a hybrid approach that performs independent sampling is difficult to reduce the total number of objective function evaluation, because the approach performs an optimization process per each point produced, contrary to most evolutionary approaches that produce a set of points in a single search process. However, with the help of some additional mechanisms, is possible to perform a similar number of objective evaluations than other approaches.

Given such conditions, the proposed hybrid approach obtained very good approximations in hard problems, while the performance in easy and regular problems is very competitive.

Finally, we present an approach for incorporating preferences of the decision maker to the multiobjective approach, for special cases when the entire Pareto front is not necessary, thus reducing the cost of the search focusing in a smaller region. With this last contribution, we conclude presenting different proposals that cover a wide range of situations.

10.1 Conclusions

The following conclusions were obtained during the development process of the approach for constrained optimization, its experimental validation, as well as its application as a part of a hybrid algorithm for multiobjective optimization.

- The proper use of domain information incorporated into an evolutionary algorithm can enhance its convergence capabilities (as is the case of the results in Chapter 5), the quality of solutions obtained (as is the case of the results in several problems in Chapter 8), or both.

- Cultural algorithms are a promising alternative to incorporate domain information into an evolutionary algorithm. The main advantage of cultural algorithms is that, in addition to the domain knowledge incorporated *a priori*, the information can be extracted from the population during the search process itself, giving characteristics of self-adaptation to different instances of the problem.

- For the case of constrained optimization with real-valued decision variables, differential evolution is a powerful alternative. Additionally to this work, recently several authors have chosen differential evolution for constrained optimization tasks [94, 105, 112].

- If we use different knowledge sources emphasizing different aspects of the search, the interaction of them is beneficial for the entire search process.

- The approach for constrained optimization presented here is able to reduce the computational cost (measured as the number of objective functions evaluations) compared with other evolutionary algorithms, when approximating the global optimum of constrained optimization problems. The experiments included in Chapter 5 reported competitive results, but requiring 7%-40% of the number of evaluations of other 4 algorithms. This is specially useful when the evaluation of objective functions are computationally expensive (*e.g.* when there is no analytical expression for the objective function, when the results are provided by a simulator, or when they require intensive computations, as is the case of the optimization of trusses adopted in this work).

- The convergence capabilities of the proposed approach are useful in several applications, including the hybridization with other techniques. We exploited this characteristic in a proposal for multiobjective optimization problems, obtaining good results on hard problems (see Chapter 8).

- Even when mathematical programming methods of independent sampling (as the ε-constraint method) are not very popular in evolutionary computation, are useful in certain situations, like the case when dealing with very difficult problems for other evolutionary approaches. This is because these methods focus on a single sample of the Pareto front at a time, and thus can achieve a better convergence towards that point.

- The hybrid methods based on independent sampling can result very expensive, because of the several optimization processes carried out. An alternative to reduce this cost is reducing the number of points produced. In any case, if a good single-objective optimizer is chosen, each point will be a good approximation. Our experiments with the technique based only on independent sampling, obtained better results in 6 out of 9 problems with different characteristics, when comparing with the results of the NSGA-II. In one case, the output sets of our proposed technique always dominate the sets of those produced by the NSGA-II.

- Another alternative to reduce that cost, is to perform a second phase

algorithm. The first stage is responsible of convergence, and the second must be designed to spread the points. We implemented a technique for that second phase, and we obtained better results in 10 out of 15 different problems, when comparing with the results of the NSGA-II. These experiments were performed with a lower number of function evaluations, when comparing with the previous case.

- The incorporation of preferences to a multiobjective algorithm is an advisable practice when the needed information is available (in this case, a vector of goals), because it can reduce the computational cost required by the algorithm. With the approach proposed here for incorporating preferences, we were able to perform experiments with problems that have up to 10 objectives. In this case, it would be useful a framework that allows comparisons of the performances different algorithms. The approach proposed here can be useful for such task, and can be used as a starting point for more sophisticated methodologies.

10.2 Future work

Cultural algorithms have been scarcely applied, considering the wide range of problems adopted for other evolutionary approaches. There exist also little theoretical studies about them. In this field, thus, there is much work to do. Particular research ideas are the following:

- The application of cultural algorithms to other optimization problems. Particularly suitable problems are those that have been intensively studied, and we have important results for them (we posses a considerable amount of domain knowledge). Such is the case of several combinatorial optimization problems, and scheduling problems, where cultural algorithms are promising alternatives.

- The development of theoretical models to describe the properties of cultural algorithms. Seminal works are in [146, 14], but no further advances have been developed.

- In constrained optimization, it may be possible to design a fifth knowledge source: the domain knowledge. This task may be difficult because of the generality of the problem; this is more like information added *a priori*. Reynolds has previously suggested that five knowledge sources (those included in this work, plus the domain knowledge) are a complete set, and other sources can be represented as a combination of this set. That statement remains to be proven.

- In multiobjective optimization, the experimentation and comparison methodologies are constantly developing. An important work that remains to be done is a good framework that allows a proper experimentation and comparison of results of approaches that incorporate preferences.

A

Single-objective Constrained Optimization Problems

The following is the benchmark originally proposed in [116] and extended in [153] which is used in this thesis. The 13 test problems are the following.

g01.

Minimize

$$f(\mathbf{x}) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i$$

subject to

$$\begin{aligned}
g_1(\mathbf{x}) &= 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0 \\
g_2(\mathbf{x}) &= 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \leq 0 \\
g_3(\mathbf{x}) &= 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0 \\
g_4(\mathbf{x}) &= -8x_1 + x_{10} \leq 0 \\
g_5(\mathbf{x}) &= -8x_2 + x_{11} \leq 0 \\
g_6(\mathbf{x}) &= -8x_3 + x_{12} \leq 0 \\
g_7(\mathbf{x}) &= -2x_4 - x_5 + x_{10} \leq 0 \\
g_8(\mathbf{x}) &= -2x_6 - x_7 + x_{11} \leq 0 \\
g_9(\mathbf{x}) &= -2x_8 - x_9 + x_{12} \leq 0
\end{aligned}$$

where: $0 \leq x_i \leq 1$ ($i = 1, \ldots, 9$), $0 \leq x_i \leq 100$ ($i = 10, 11, 12$) and $0 \leq x_{13} \leq 1$.

The optimum solution is $\mathbf{x}^* = (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)$ where $f(\mathbf{x}^*) = -15$.

g02.

Maximize

$$f(\mathbf{x}) = \left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|$$

subject to

$$\begin{aligned} g_1(\mathbf{x}) &= 0.75 - \prod_{i=1}^{n} x_i \leq 0 \\ g_2(\mathbf{x}) &= \sum_{i=1}^{n} x_i - 7.5n \leq 0 \end{aligned}$$

where: $n = 20$ y $0 \leq x_i \leq 10$ ($i = 1, \ldots, n$).

The best known solution is $f(\mathbf{x}^*) = 0.803619$.

g03.

Maximize

$$f(\mathbf{x}) = (\sqrt{n})^n \prod_{i=1}^{n} x_i$$

subject to

$$h_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 - 1 = 0$$

where: $n = 10$ y $0 \leq x_i \leq 1$ ($i = 1, \ldots, n$).

The optimum solution is $x_i^* = 1/\sqrt{n}$ ($i = 1, \ldots, n$) where $f(\mathbf{x}^*) = 1$.

g04.

Minimize

$$f(\mathbf{x}) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141$$

subject to

$$\begin{aligned}
g_1(\mathbf{x}) &= 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 \\
&\quad -0.0022053x_3x_5 - 92 \leq 0 \\
g_2(\mathbf{x}) &= -85.334407 - 0.0056858x_2x_5 - 0.0006262x_1x_4 \\
&\quad +0.0022053x_3x_5 \leq 0 \\
g_3(\mathbf{x}) &= 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 \\
&\quad +0.0021813x_3^2 - 110 \leq 0 \\
g_4(\mathbf{x}) &= -80.51249 - 0.0071317x_2x_5 - 0.0029955x_1x_2 \\
&\quad -0.0021813x_3^2 + 90 \leq 0 \\
g_5(\mathbf{x}) &= 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 \\
&\quad +0.0019085x_3x_4 - 25 \leq 0 \\
g_6(\mathbf{x}) &= -9.300961 - 0.0047026x_3x_5 - 0.0012547x_1x_3 \\
&\quad -0.0019085x_3x_4 + 20 \leq 0
\end{aligned}$$

where: $78 \leq x_1 \leq 102$, $33 \leq x_2 \leq 45$, $27 \leq x_i \leq 45$ ($i = 3, 4, 5$).

The optimum solution is $\mathbf{x}^* = (78, 33, 29.995256025682, 45, 36.775812905788)$ where $f(\mathbf{x}^*) = -30665.539$.

g05.

Minimize

$$f(\mathbf{x}) = 3x_1 + 0.000001x_1^3 + 2x_2 + (0.000002/3)x_2^3$$

subject to

$$\begin{aligned}
g_1(\mathbf{x}) &= -x_4 + x_3 - 0.55 \leq 0 \\
g_2(\mathbf{x}) &= -x_3 + x_4 - 0.55 \leq 0 \\
h_3(\mathbf{x}) &= 1000\sin(-x_3 - 0.25) + 1000\sin(-x_4 - 0.25) \\
&\quad +894.8 - x_1 = 0
\end{aligned}$$

$$\begin{aligned} h_4(\mathbf{x}) &= 1000\sin(x_3 - 0.25) + 1000\sin(x_3 - x_4 - 0.25) \\ &\quad +894.8 - x_2 = 0 \\ h_5(\mathbf{x}) &= 1000\sin(x_4 - 0.25) + 1000\sin(x_4 - x_3 - 0.25) \\ &\quad +1294.8 = 0 \end{aligned}$$

where: $0 \leq x_1 \leq 1200$, $0 \leq x_2 \leq 1200$, $-0.55 \leq x_3 \leq 0.55$ and $-0.55 \leq x_4 \leq 0.55$.

The best known solution is $\mathbf{x}^* = (679.9453, 1026.067, 0.1188764, -0.3962336)$ where $f(\mathbf{x}^*) = 5126.4981$.

g06.

Minimize

$$f(\mathbf{x}) = (x_1 - 10)^3 + (x_2 - 20)^3$$

subject to

$$\begin{aligned} g_1(\mathbf{x}) &= -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0 \\ g_2(\mathbf{x}) &= (x_1 - 6)^2 + (x_2 - 5) - 82.81 \leq 0 \end{aligned}$$

where: $13 \leq x_1 \leq 100$ and $0 \leq x_2 \leq 100$.

The optimum solution is $\mathbf{x}^* = (14.095, 0.84296)$ where $f(\mathbf{x}^*) = -6961.81388$.

g07.

Minimize

$$\begin{aligned} f(\mathbf{x}) &= x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 \\ &\quad +4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 \\ &\quad +7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45 \end{aligned}$$

subject to

$$\begin{aligned} g_1(\mathbf{x}) &= -105 + 4x_1 + 5x_2 - 3x_7 + 9x_8 \leq 0 \\ g_2(\mathbf{x}) &= 10x_1 - 8x_2 - 17x_7 + 2x_8 \leq 0 \\ g_3(\mathbf{x}) &= -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \leq 0 \\ g_4(\mathbf{x}) &= 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^2 - 7x_4 - 120 \leq 0 \end{aligned}$$

$$
\begin{aligned}
g_5(\mathbf{x}) &= 5x_1^2 + 8x_2 + (x_3 - 6)^2 - 2x_4 - 40 \leq 0 \\
g_6(\mathbf{x}) &= x_1^2 + 2(x_2 - 2)^2 - 2x_1x_2 + 14x_5 - 6x_6 \leq 0 \\
g_7(\mathbf{x}) &= 0.5(x_1 - 8)^2 + 2(x_2 - 4)^2 + 3x_5^2 - x_6 - 30 \leq 0 \\
g_8(\mathbf{x}) &= -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_{10} \leq 0
\end{aligned}
$$

where: $-10 \leq x_i \leq 10$ ($i = 1, \ldots, 10$).

The optimum solution is $\mathbf{x}^* = (2.171996, 2.363683, 8.773926, 5.095984, 0.9906548, 1.430574, 1.321644, 9.828726, 8.280092, 8.375927)$ where $f(\mathbf{x}^*) = 24.3062091$.

g08.

Maximize

$$
f(\mathbf{x}) = \frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1 + x_2)}
$$

subject to

$$
\begin{aligned}
g_1(\mathbf{x}) &= x_1^2 - x_2 + 1 \leq 0 \\
g_2(\mathbf{x}) &= 1 - x_1 + (x_2 - 4)^2 \leq 0
\end{aligned}
$$

where: $0 \leq x_1 \leq 10$, $0 \leq x_2 \leq 10$.

The optimum solution is $\mathbf{x}^* = (1.2279713, 4.2453733)$ where $f(\mathbf{x}^*) = 0.095825$.

g09.

Minimize

$$
\begin{aligned}
f(\mathbf{x}) &= (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 \\
&\quad +7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7
\end{aligned}
$$

subject to

$$
\begin{aligned}
g_1(\mathbf{x}) &= -127 + 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 \leq 0 \\
g_2(\mathbf{x}) &= -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 \leq 0 \\
g_3(\mathbf{x}) &= -196 + 23x_1 + x_2^2 + 6x_6^2 - 8x_7 \leq 0 \\
g_4(\mathbf{x}) &= 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \leq 0
\end{aligned}
$$

where: $-10 \leq x_i \leq 10$ ($i = 1, \ldots, 7$).

The optimum solution is $\mathbf{x}^* = (2.330499, 1.951372, -0.4775414, 4.365726, -0.6244870, 1.038131, 1.594227)$ where $f(\mathbf{x}^*) = 680.6300573$.

g10.

Minimize

$$f(\mathbf{x}) = x_1 + x_2 + x_3$$

subject to

$$\begin{aligned}
g_1(\mathbf{x}) &= -1 + 0.0025(x_4 + x_6) \leq 0 \\
g_2(\mathbf{x}) &= -1 + 0.0025(x_5 + x_7 - x_4) \leq 0 \\
g_3(\mathbf{x}) &= -1 + 0.01(x_8 - x_5) \leq 0 \\
g_4(\mathbf{x}) &= -x_1 x_6 + 833.33252 x_4 + 100 x_1 - 83333.333 \leq 0 \\
g_5(\mathbf{x}) &= -x_2 x_7 + 1250 x_5 + x_2 x_4 - 1250 x_4 \leq 0 \\
g_6(\mathbf{x}) &= -x_3 x_8 + 1250000 + x_3 x_5 - 2500 x_5 \leq 0
\end{aligned}$$

where: $100 \leq x_1 \leq 10000$, $1000 \leq x_i \leq 10000$ ($i = 2, 3$) and $10 \leq x_i \leq 1000$ ($i = 4, \ldots, 8$).

The optimum solution is $\mathbf{x}^* = (579.19, 1360.13, 5109.92, 182.0174, 295.5985, 217.9799, 286.40, 395.5979)$, where $f(\mathbf{x}^*) = 7049.25$.

g11.

Minimize

$$f(\mathbf{x}) = x_1^2 + (x_2 - 1)^2$$

subject to

$$h(\mathbf{x}) = x_2 - x_1^2 = 0$$

where: $-1 \leq x_1 \leq 1$, $-1 \leq x_2 \leq 1$.

The optimum solution is $\mathbf{x}^* = (\pm 1/\sqrt{2}, 1/2)$ where $f(\mathbf{x}^*) = 0.75$.

g12.

Maximize

$$f(\mathbf{x}) = \frac{(100 - (x_1 - 5)^2 - (x_2 - 5)^2 - (x_3 - 5)^2)}{100}$$

subject to

$$g(\mathbf{x}) \;=\; (x_1 - p)^2 + (x_2 - q)^2 + (x_3 - r)^2 - 0.0625 \leq 0$$

where: $0 \leq x_i \leq 10$ $(i = 1, 2, 3)$, and $p, q, r = 1, 2, \ldots, 9$.

The optimum solution is $\mathbf{x}^* = (5, 5, 5)$ where $f(\mathbf{x}^*) = 1$.

g13.

Minimize

$$f(\mathbf{x}) = e^{x_1 x_2 x_3 x_4 x_5}$$

subject to

$$\begin{aligned} h_1(\mathbf{x}) &= x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0 \\ h_2(\mathbf{x}) &= x_2 x_3 - 5 x_4 x_5 = 0 \\ h_3(\mathbf{x}) &= x_1^3 + x_2^3 - 1 = 0 \end{aligned}$$

where: $-2.3 \leq x_i \leq 2.3$ $(i = 1, 2)$ and $-3.2 \leq x_i \leq 3.2$ $(i = 3, 4, 5)$.

The optimum solution is $\mathbf{x}^* = (-1.717143, 1.595709, 1.827247, -0.7636413, -0.763645)$ where $f(\mathbf{x}^*) = 0.0539498$.

Additionally to this standard benchmark, we adopted in this thesis the optimization of a 10-bar plane truss and a 200-bar plane truss.

Design of a 10-bar plane truss.

Consider the 10-bar plane truss shown in Figure A.1 [5]. The problem is to find the cross-sectional area of each member of this truss, such that we minimize its weight. The problem is subject to both displacement and stress constraints. The weight of the truss is given by $f(\mathbf{x})$.

$$f(\mathbf{x}) = \sum_{j=1}^{10} \rho \, A_j \, L_j$$

where x is the candidate solution, A_j is the cross-sectional area of the jth member, L_j is the length of the jth member, and ρ is the weight density of the material. The assumed data are: modulus of elasticity,

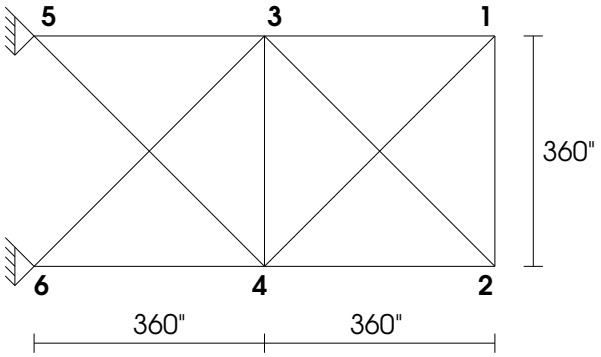


Figure A.1: 10-bar plane truss adopted for problem **G14**.

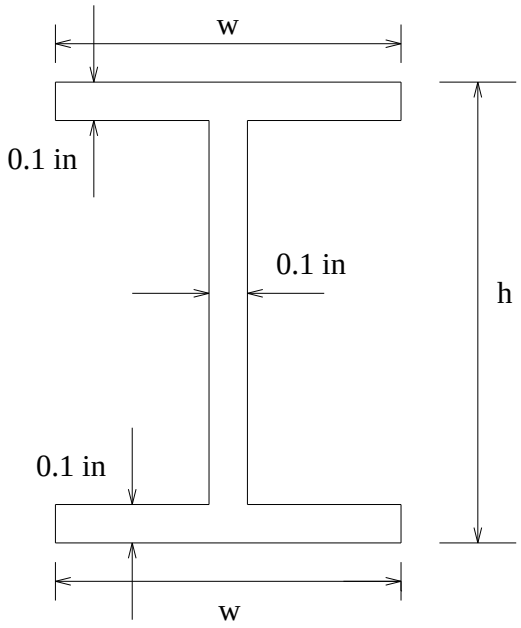


Figure A.2: Cross-section used for the 10-bar plane truss from problem **G14**.

$E = 1.09 \times 10^4$ ksi, $\rho = 0.10$ lb/in^3, and a load of 100 kips in the negative y-direction is applied at nodes 2 and 4. The maximum allowable stress of each member is called σ_a, and it is assumed to be ± 25 ksi. The maximum allowable displacement of each node (horizontal and vertical) is represented by u_a, and is assumed to be 2 inches. The minimum allowable cross-section area is 0.10 in^2 for all members. The cross-section of each element can be different, and is defined by the I-section shown in Figure A.2, with the depth and width as design variables. The web thickness and flange thickness are each kept fixed at 0.1 in. The problem has, therefore, 20 design variables.

To solve this problem, it was necessary to add a module responsible for the analysis of the plane truss. This module uses the stiffness method [55] to analyze the structure, and returns the values of the stress and displacement constraints, as well as the total weight of the structure.

Design of a 200-bar plane truss.

Consider the 200-bar plane truss taken from Belegundu [5] and shown in Figure A.3. The problem is to find the cross-sectional area of each member of this truss, such that we minimize its weight. The problem is subject to both displacement and stress constraints.

There are a total of three loading conditions: (1) 1 kip acting in positive x-direction at node points 1, 6, 15, 20, 29, 34, 43, 48, 57, 62, and 71; (2) 10 kips acting in negative y-direction at node points 1, 2, 3, 4, 5, 6, 8, 10, 12, 14, 15, 16, 17, 18, 19, 20, 24, 71, 72, 73, 74, and 75; and (3) loading condition 1 and 2 acting together. The 200 elements of this truss linked to 29 groups. The grouping information is shown in Table A.1. The stress in each element is limited to a value of 10 ksi for both tension and compression members. Young's modulus of elasticity = 30,000 ksi, weight density = 0.283×10^{-3} kips/in^3.

Group Number	Member Number
1	1,2,3,4
2	5,8,11,14,17
3	19,20,21,22,23,24
4	18,25,56,63,94,101,132,139,170,177
5	26,29,32,35,38
6	6,7,9,10,12,13,15,16,27,28,30,31,33,34,36,37
7	39,40,41,42
8	43,46,49,52,55
9	57,58,59,60,61,62
10	64,67,70,73,76
11	44,45,47,48,50,51,53,54,65,66,68,69,71,72,74,75
12	77,78,79,80
13	81,84,87,90,93
14	95,96,97,98,99,100
15	102,105,108,111,114
16	82,83,85,86,88,89,91,92,103,104,106,107,109,110,112,113
17	115,116,117,118
18	119,122,125,128,131
19	133,134,135,136,137,138
20	140,143,146,149,152
21	120,121,123,124,126,127,129,130,141,142,144,145,147,148,150,151
22	153,154,155,156
23	157,160,163,166,169
24	171,172,173,174,175,176
25	178,181,184,187,190
26	158,159,161,162,164,165,167,168,179,180,182,183,185,186,188,189
27	191,192,193,194
28	195,197,198,200
29	196,199

Table A.1: Group membership for the 200-bar plane truss from problem **G15**.

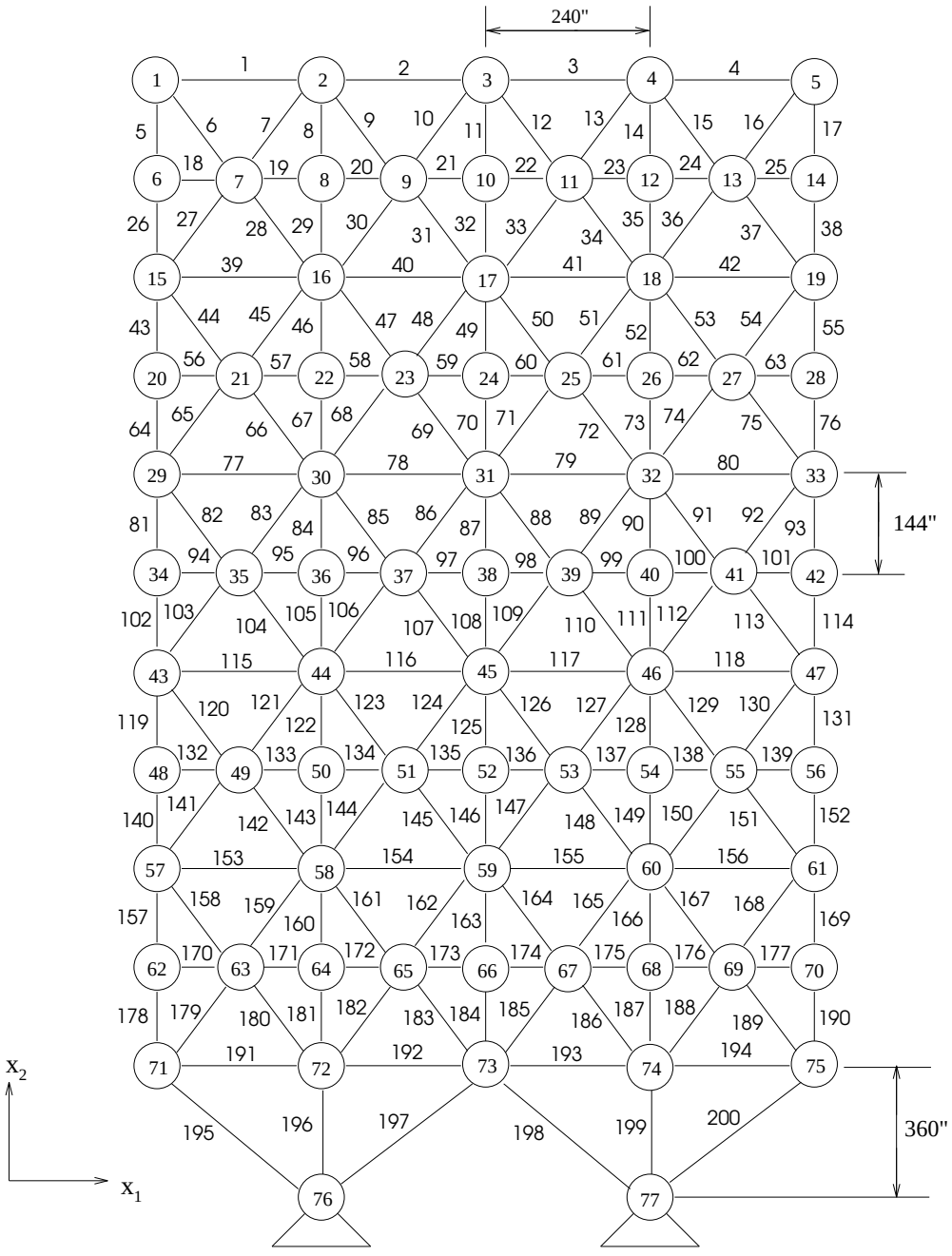


Figure A.3: 200-bar plane truss used for problem **G15**.

B

Multiobjective Optimization Problems

The first test suite adopted in this thesis is the one from [71]. The authors of this suite propose a set of transformations that are sequentially applied to the decison variables, where each transformation adds a desired characteristic to the problem. All the problems generated with this methodology follow this format:

$$
\begin{aligned}
\text{Given } \mathbf{z} &= [z_1, \ldots, z_k, z_{k+1}, \ldots, z_n] \\
\text{Minimize } f_{m=1:M}(\mathbf{x}) &= Dx_M + S_m h_m(x_1, \ldots, x_{M-1}) \\
\text{where } \mathbf{x} &= [x_1, \ldots, x_M] \\
&= [\max(t^p_M, A_1)(t^p_1 - 0.5) + 0.5, \ldots, \\
&\quad \max(t^p_M, A_{M-1})(t^p_{M-1} - 0.5) + 0.5, t^p_M] \\
\mathbf{t}^p &= [t^p_1, \ldots, t^p_M] \leftarrow\!\shortmid \mathbf{t}^{p-1} \leftarrow\!\shortmid \ldots \leftarrow\!\shortmid \mathbf{t}^1 \leftarrow\!\shortmid \mathbf{z}_{[0,1]} \\
\mathbf{z}_{[0,1]} &= [z_{1,[0,1]}, \ldots, z_{n,[0,1]}] \\
&= [z_1/z_{1,\max}, \ldots, z_n/z_{n,\max}]
\end{aligned}
$$

where $\mathbf{z}$ is the vector of decision variables with $0 \leq z_i \leq z_{i,\max}$, D, $A_{1:M-1}$ and $S_{1:M}$ are constants to modify position and scale of the Pareto front.

The $h_{1:M}$ functions define the shape of the Pareto front, which can be linear, convex, concave, mixed convex and concave, and disconnected.

Such characteristics are obtained with the following set of functions:

$$\begin{aligned}
\text{linear}_1(x_1,\ldots,x_{M-1}) &= \prod_{i=1}^{M-1} x_i \\
\text{linear}_{m=2:M-1}(x_1,\ldots,x_{M-1}) &= \left(\prod_{i=1}^{M-m} x_i\right)(1-x_{M-m+1}) \\
\text{linear}_M(x_1,\ldots,x_{M-1}) &= 1-x_1
\end{aligned}$$

$$\begin{aligned}
\text{convex}_1(x_1,\ldots,x_{M-1}) &= \prod_{i=1}^{M-1}(1-\cos(x_i\pi/2)) \\
\text{convex}_{m=2:M-1}(x_1,\ldots,x_{M-1}) &= \left(\prod_{i=1}^{M-m}(1-\cos(x_i\pi/2))\right)\cdot \\
&\quad (1-\sin(x_{M-m+1}\pi/2)) \\
\text{convex}_M(x_1,\ldots,x_{M-1}) &= 1-\sin(x_1\pi/2)
\end{aligned}$$

$$\begin{aligned}
\text{concave}_1(x_1,\ldots,x_{M-1}) &= \prod_{i=1}^{M-1}\sin(x_i\pi/2)) \\
\text{concave}_{m=2:M-1}(x_1,\ldots,x_{M-1}) &= \left(\prod_{i=1}^{M-m}\sin(x_i\pi/2)\right)\cdot \\
&\quad \cos(x_{M-m+1}\pi/2) \\
\text{concave}_M(x_1,\ldots,x_{M-1}) &= \cos(x_1\pi/2)
\end{aligned}$$

$$\text{mixed}_M(x_1,\ldots,x_{M-1}) = \left(1-x_1-\frac{\cos(2A\pi x_1+\pi/2)}{2A\pi}\right)^{\alpha}$$

$$\text{disc}_M(x_1,\ldots,x_{M-1}) = 1-x_1^{\alpha}\cos^2(Ax_1^{\beta}\pi)$$

For a mixed Pareto front, when $\alpha > 1$ the overall shape is concave, when $\alpha < 1$ it is concave, and when $\alpha = 1$ it is linear; the number of segments concave-convex is A. For a disconnected Pareto front, α controlls the overall shape in the same way it do for a mixed front; β controlls the location of the disconnected segments, and the number of disconnected segments is A.

The rest of the characteristics of the problem are added through a set of transformations. Huband et al. distinguish between three types of transformations, based on the characteristics they emphasize as important when designing multiobjective problems. Bias transformations produce a bias in the fitness landscape, and are used to produce polynomial bias, flat regions, or other type of bias depending of the values of another variables; shift transformations move the location of optimal values, and are used to apply a linear shift, or to produce deceptive and multimodal problems; and reduction transformations, which combine the values of several variables into a single one, this is used to produce nonseparability of the problem (dependency between variables). The set of transformations is the following:

$$\text{b\_poly}(y, \alpha) = y^{\alpha}$$

$$\begin{aligned}\text{b\_flat}(y, A, B, C) &= A + \min(0, \lfloor y - B \rfloor)\frac{A(B-y)}{B} - \\ &\quad \min(0, \lfloor C - y \rfloor)\frac{(1-A)(y-C)}{1-C}\end{aligned}$$

$$\text{b\_param}(y, u(\mathbf{y}'), A, B, C) = y^{B+(C-B)(A-(1-2u(\mathbf{y}'))|\lfloor 0.5-u(\mathbf{y}') \rfloor + A|)}$$

$$\text{s\_linear}(y, A) = \frac{|y-A|}{|\lfloor A - y \rfloor + A|}$$

$$\begin{aligned}\text{s\_decept}(y, A, B, C) &= 1 + (|y-A| - B)\left(\frac{\lfloor y - A + B \rfloor \left(1 - C + \frac{A-B}{B}\right)}{A-B} + \right. \\ &\quad \left. \frac{\lfloor A + B - y \rfloor \left(1 - C + \frac{1-A-B}{B}\right)}{1-A-B} + \frac{1}{B}\right)\end{aligned}$$

$$\begin{aligned}\text{s\_multi}(y, A, B, C) &= \left(1 + \cos\left((4A+2)\pi\left(0.5 - \frac{|y-C|}{2(\lfloor C-y \rfloor + C)}\right)\right) + \right. \\ &\quad \left. 4B\left(\frac{|y-C|}{2(\lfloor C-y \rfloor + C)}\right)^2\right) \Big/ (b+2)\end{aligned}$$

$$\text{r\_sum}(\mathbf{y}, \mathbf{w}) = \frac{\sum_{i=1}^{|y|} w_1 y_i}{\sum_{i=1}^{|\mathbf{y}|} w_i}$$

$$\text{r\_nonsep}(\mathbf{y}, A) = \frac{\sum_{j=1}^{|\mathbf{y}|} \left(y_j + \sum_{k=0}^{A-2} \left| y_j - y_{1+(j+k) mod |\mathbf{y}|} \right| \right)}{\frac{|\mathbf{y}|}{A} \left\lceil \frac{A}{2} \right\rceil \left(1 + 2A - 2 \left\lceil \frac{A}{2} \right\rceil \right)}$$

With this set of transformations and shape functions, Huband et al. propose a set of scalable problemas. It is easy to obtain the characteristics of each based on the transformation and shape functions used.

WFG1

Minimize

$$\begin{aligned} f_{m=1:M-1}(\mathbf{x}) &= x_M + S_m \text{convex}_m(x_1, \ldots, x_{M-1}) \\ f_M(\mathbf{x}) &= x_M + S_M \text{mixed}_M(x_1, \ldots, x_{M-1}) \end{aligned}$$

where

$$\begin{aligned} y_{i=1:M-1} &= \text{r\_sum}([y'_{(i-1)k/(M-1)+1}, \ldots, y'_{ik/(M-1)}], \\ &\quad [2((i-1)k/(M-1)+1), \ldots, 2ik/(M-1)]) \\ y_M &= \text{r\_sum}([y'_{k+1}, \ldots, y'_n], [2(k+1), \ldots, 2n]) \\ y'_{i=1:n} &= \text{b\_poly}(y''_i, 0.02) \\ y''_{i=1:k} &= y'''_i \\ y''_{i=k+1:n} &= \text{b\_flat}(y'''_i, 0.8, 0.75, 0.85) \\ y'''_{i=1:k} &= z_{i,[0,1]} \\ y'''_{i=k+1:n} &= \text{s\_linear}(z_{i,[0,1]}, 0.35) \end{aligned}$$

WFG2

Minimize

$$\begin{aligned} f_{m=1:M-1}(\mathbf{x}) &= x_M + S_m \text{convex}_m(x_1, \ldots, x_{M-1}) \\ f_M(\mathbf{x}) &= x_M + S_M \text{disc}_M(x_1, \ldots, x_{M-1}) \end{aligned}$$

where

$$\begin{aligned} y_{i=1:M-1} &= \text{r\_sum}([y'_{(i-1)k/(M-1)+1}, \ldots, y'_{ik/(M-1)}], [1, \ldots, 1]) \\ y_M &= \text{r\_sum}([y'_{k+1}, \ldots, y'_{k+l/2}], [1, \ldots, 1]) \\ y'_{i=1:k} &= y''_i \\ y'_{i=k+1:k+l/2} &= \text{r\_nonsep}([y''_{k+2(i-k)-1}, y''_{k+2(i-k)}], 2) \\ y''_{i=1:k} &= z_{i,[0,1]} \\ y''_{i=k+1:n} &= \text{s\_linear}(z_{i,[0,1]}, 0.35) \end{aligned}$$

WFG3

Minimize

$$f_{m=1:M}(\mathbf{x}) \quad = \quad x_M + S_m \text{linear}_m(x_1, \ldots, x_{M-1})$$

where

$$\begin{aligned}
y_{i=1:M-1} &= \text{r_sum}([y'_{(i-1)k/(M-1)+1}, \ldots, y'_{ik/(M-1)}], [1, \ldots, 1]) \\
y_M &= \text{r_sum}([y'_{k+1}, \ldots, y'_{k+l/2}], [1, \ldots, 1]) \\
y'_{i=1:k} &= y''_i \\
y'_{i=k+1:k+l/2} &= \text{r_nonsep}([y''_{k+2(i-k)-1}, y''_{k+2(i-k)}], 2) \\
y''_{i=1:k} &= z_{i,[0,1]} \\
y''_{i=k+1:n} &= \text{s_linear}(z_{i,[0,1]}, 0.35)
\end{aligned}$$

WFG4

Minimize

$$f_{m=1:M}(\mathbf{x}) \quad = \quad x_M + S_m \text{concave}_m(x_1, \ldots, x_{M-1})$$

where

$$\begin{aligned}
y_{i=1:M-1} &= \text{r_sum}([y'_{(i-1)k/(M-1)+1}, \ldots, y'_{ik/(M-1)}], [1, \ldots, 1]) \\
y_M &= \text{r_sum}([y'_{k+1}, \ldots, y'_n], [1, \ldots, 1]) \\
y'_{i=1:n} &= \text{s_multi}(z_{i,[0,1]}, 30, 10, 0.35)
\end{aligned}$$

WFG5

Minimize

$$f_{m=1:M}(\mathbf{x}) \quad = \quad x_M + S_m \text{concave}_m(x_1, \ldots, x_{M-1})$$

where

$$\begin{aligned}
y_{i=1:M-1} &= \text{r_sum}([y'_{(i-1)k/(M-1)+1}, \ldots, y'_{ik/(M-1)}], [1, \ldots, 1]) \\
y_M &= \text{r_sum}([y'_{k+1}, \ldots, y'_n], [1, \ldots, 1]) \\
y'_{i=1:n} &= \text{s_decept}(z_{i,[0,1]}, 0.35, 0.001, 0.05)
\end{aligned}$$

WFG6

Minimize

$$f_{m=1:M}(\mathbf{x}) = x_M + S_m \text{concave}_m(x_1, \ldots, x_{M-1})$$

where

$$\begin{aligned}
y_{i=1:M-1} &= \text{r_nonsep}([y'_{(i-1)k/(M-1)+1}, \ldots, y'_{ik/(M-1)}], k/(M-1)) \\
y_M &= \text{r_nonsep}([y'_{k+1}, \ldots, y'_n], l) \\
y'_{i=1:k} &= z_{i,[0,1]} \\
y'_{i=k+1:n} &= \text{s_linear}(z_{i,[0,1]}, 0.35)
\end{aligned}$$

WFG7

Minimize

$$f_{m=1:M}(\mathbf{x}) = x_M + S_m \text{concave}_m(x_1, \ldots, x_{M-1})$$

where

$$\begin{aligned}
y_{i=1:M-1} &= \text{r_sum}([y'_{(i-1)k/(M-1)+1}, \ldots, y'_{ik/(M-1)}], [1, \ldots, 1]) \\
y_M &= \text{r_sum}([y'_{k+1}, \ldots, y'_n], [1, \ldots, 1]) \\
y'_{i=1:k} &= y''_i \\
y'_{i=k+1:n} &= \text{s_linear}(y''_i, 0.35) \\
y''_{i=1:k} &= \text{b_param}(z_{i,[0,1]}, \text{r_sum}([z_{i+1,[0,1]}, \ldots, z_{n,[0,1]}], \\
&\quad [1, \ldots, 1]), 0.98/49.98, 0.02, 50) \\
y''_{i=k+1:n} &= z_{i,[0,1]}
\end{aligned}$$

WFG8

Minimize

$$f_{m=1:M}(\mathbf{x}) = x_M + S_m \text{concave}_m(x_1, \ldots, x_{M-1})$$

where

$$\begin{aligned}
y_{i=1:M-1} &= \text{r_sum}([y'_{(i-1)k/(M-1)+1}, \ldots, y'_{ik/(M-1)}], [1, \ldots, 1]) \\
y_M &= \text{r_sum}([y'_{k+1}, \ldots, y'_n], [1, \ldots, 1])
\end{aligned}$$

$$
\begin{aligned}
y'_{i=1:k} &= y''_i \\
y'_{i=k+1:n} &= \text{s\_linear}(y''_i, 0.35) \\
y''_{i=1:k} &= z_{i,[0,1]} \\
y''_{i=k+1:n} &= \text{b\_param}(z_{i,[0,1]}, \text{r\_sum}([z_{1,[0,1]}, \ldots, z_{i-1,[0,1]}], \\
&\quad [1, \ldots, 1]), 0.98/49.98, 0.02, 50)
\end{aligned}
$$

WFG9

Minimize

$$
f_{m=1:M}(\mathbf{x}) = x_M + S_m \text{concave}_m(x_1, \ldots, x_{M-1})
$$

where

$$
\begin{aligned}
y_{i=1:M-1} &= \text{r\_nonsep}([y'_{(i-1)k/(M-1)+1}, \ldots, y'_{ik/(M-1)}], k/(M-1)) \\
y_M &= \text{r\_nonsep}([y'_{k+1}, \ldots, y'_n], l) \\
y'_{i=1:k} &= \text{s\_decept}(y''_i, 0.35, 0.001, 0.05) \\
y'_{i=k+1:n} &= \text{s\_multi}(y''_i, 30, 95, 0.35) \\
y''_{i=1:n-1} &= \text{b\_param}(z_{i,[0,1]}, \text{r\_sum}([z_{i+1,[0,1]}, \ldots, z_{n,[0,1]}], \\
&\quad [1, \ldots, 1]), 0.98/49.98, 0.02, 50) \\
y''_n &= z_{n,[0,1]}
\end{aligned}
$$

The authors of this suite suggest to use this problems with 24 variables ($k = 4$ and $n = 24$). The following constants hold for all the problems:

$$
\begin{aligned}
z_{i=1:n,\max} &= 2i \\
S_{m=1:M} &= 2m \\
A_1 &= 1 \\
A_{2:M-1} &= \begin{cases} 0, & \text{for WFG3} \\ 1, & \text{otherwise} \end{cases} \\
D &= 1
\end{aligned}
$$

The second test suite adopted is the one from [125]. The problems are not scalable, and the expressions are the following.

OKA1

Minimize

$$\begin{aligned} f_1(\mathbf{x}) &= \cos\left(\frac{\pi}{12}\right) x_1 - \sin\left(\frac{\pi}{12}\right) x_2 \\ f_2(\mathbf{x}) &= \sqrt{2\pi} - \sqrt{\left|\cos\left(\frac{\pi}{12}\right) x_1 - \sin\left(\frac{\pi}{12}\right) x_2\right|} \\ &\quad +2\left|\sin\left(\frac{\pi}{12}\right) x_1 + \cos\left(\frac{\pi}{12}\right) x_2\right. \\ &\quad \left. -3\cos\left(\cos\left(\frac{\pi}{12}\right) x_1 - \sin\left(\frac{\pi}{12}\right) x_2\right) - 3\right|^{\frac{1}{3}} \end{aligned}$$

where $6\sin\left(\frac{\pi}{12}\right) \leq x_1 \leq 6\sin\left(\frac{\pi}{12}\right) + 2\pi\cos\left(\frac{\pi}{12}\right)$ and $-2\pi\sin\left(\frac{\pi}{12}\right) \leq x_2 \leq 6\cos\left(\frac{\pi}{12}\right)$.

OKA2

Minimize

$$\begin{aligned} f_1(\mathbf{x}) &= x_1 \\ f_2(\mathbf{x}) &= 1 - \frac{1}{4\pi^2}(x_1+\pi)^2 + |x_2 - 5\cos(x_1)|^{\frac{1}{3}} + |x_3 - 5\sin(x_1)|^{\frac{1}{3}} \end{aligned}$$

where $-\pi \leq x_1 \leq \pi$ and $-5 \leq x_2, x_3 \leq 5$.

Finally, we also adopted the test suite from [178]. All the problems are based in this format:

Minimize

$$\mathbf{f}(\mathbf{x}) = (f_1(x_1), f_2(\mathbf{x}))$$

with

$$f_2(\mathbf{x}) = g(x_2, \ldots, x_n)h(f_1(x_1), g(x_2, \ldots, x_n))$$

A particular instance is constructed by defining f_1, g and h. The following problems are proposed by the authors of this suite.

ZDT1

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2, \ldots, x_n) &= 1 + 9\sum_{i=2}^{n} \frac{x_i}{m-1} \\ h(f_1, g) &= 1 - \sqrt{\frac{f_1}{g}} \end{aligned}$$

where $n = 30$ and $0 \leq x_i \leq 1$.

ZDT2

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2, \ldots, x_n) &= 1 + 9 \sum_{i=2}^{n} \frac{x_i}{m-1} \\ h(f_1, g) &= 1 - \left(\frac{f_1}{g}\right)^2 \end{aligned}$$

where $n = 30$ and $0 \leq x_i \leq 1$.

ZDT3

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2, \ldots, x_n) &= 1 + 9 \sum_{i=2}^{n} \frac{x_i}{m-1} \\ h(f_1, g) &= 1 - \sqrt{\frac{f_1}{g}} - \frac{f_1}{g} \sin(10\pi f_1) \end{aligned}$$

where $n = 30$ and $0 \leq x_i \leq 1$.

ZDT4

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2, \ldots, x_n) &= 1 + 10(m-1) + \sum_{i=2}^{n} (x_i^2 - 10\cos(4\pi x_i)) \\ h(f_1, g) &= 1 - \sqrt{\frac{f_1}{g}} \end{aligned}$$

where $n = 10$, $0 \leq x_1 \leq 1$ and $-5 \leq x_{i=2,\ldots,n} \leq 5$.

C

Histograms of the Bootstrap Distributions for the Constrained Optimization Problems

In this appendix, we show the histograms of the bootstrap distributions of 1,000 resamples, for each problem of the benchmark for constrained optimization which has been produced at least two different results. The histograms are shown in Figures C.1 to C.8.

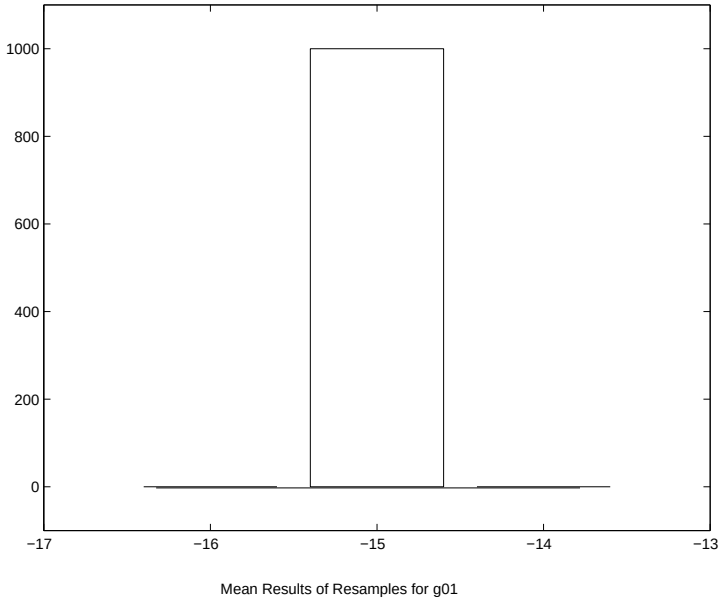


Figure C.1: Bootstrap distribution for the mean statistic for problem g01.

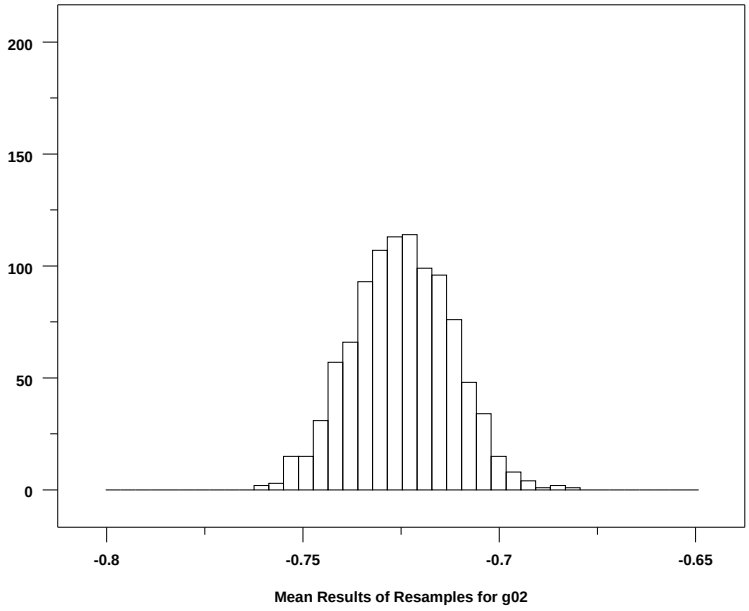


Figure C.2: Bootstrap distribution for the mean statistic for problem g02.

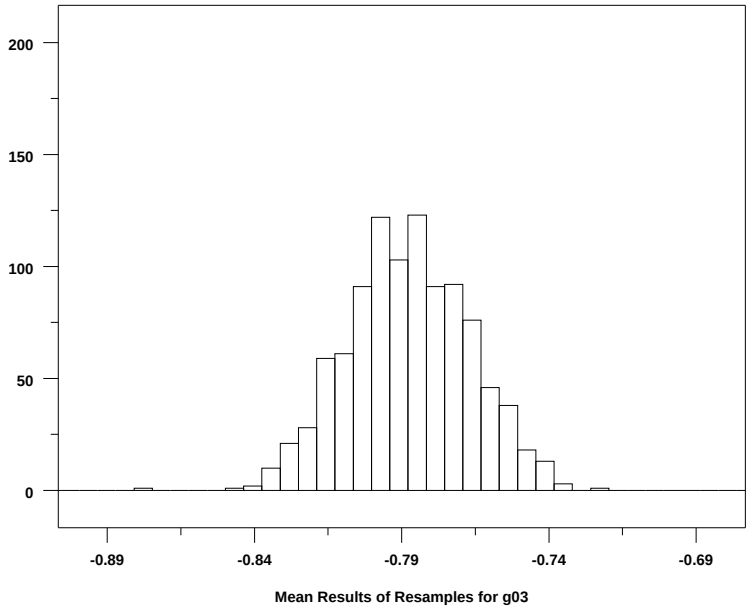


Figure C.3: Bootstrap distribution for the mean statistic for problem g03.

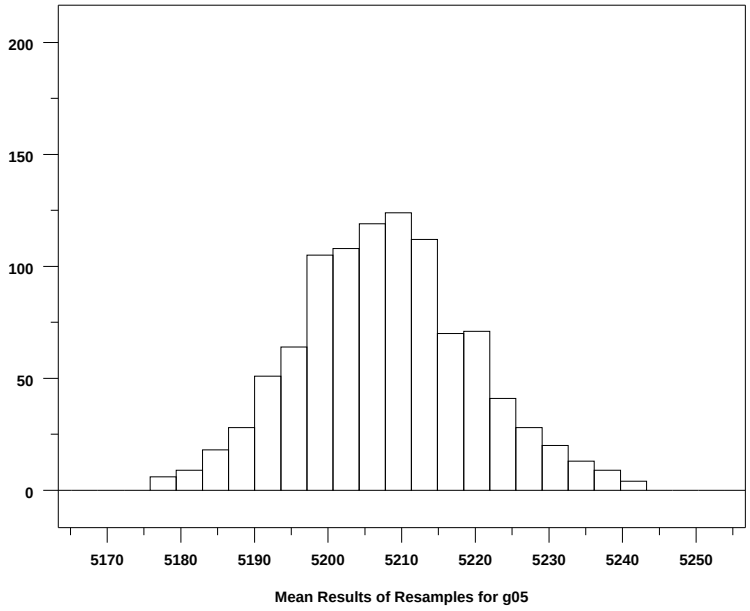


Figure C.4: Bootstrap distribution for the mean statistic for problem g05.

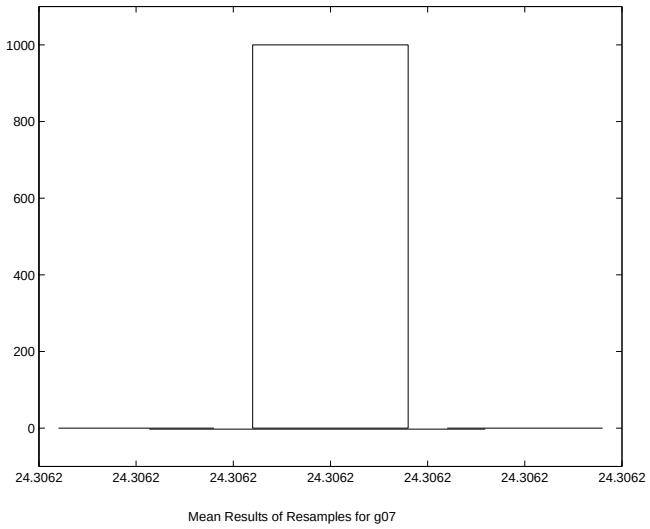


Figure C.5: Bootstrap distribution for the mean statistic for problem g07.

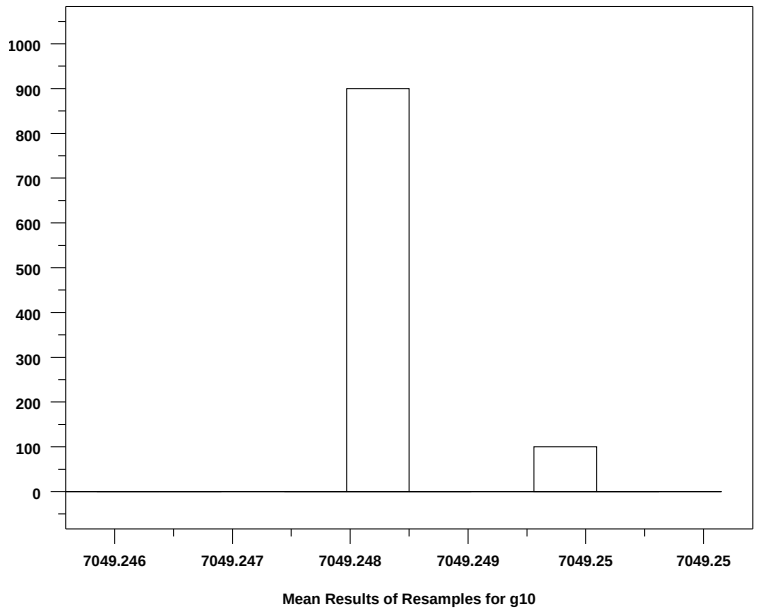


Figure C.6: Bootstrap distribution for the mean statistic for problem g10.

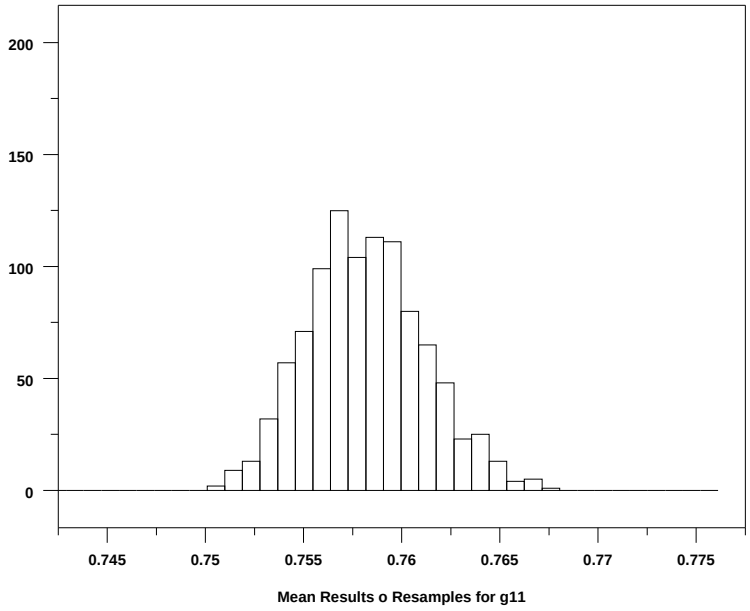


Figure C.7: Bootstrap distribution for the mean statistic for problem g11.

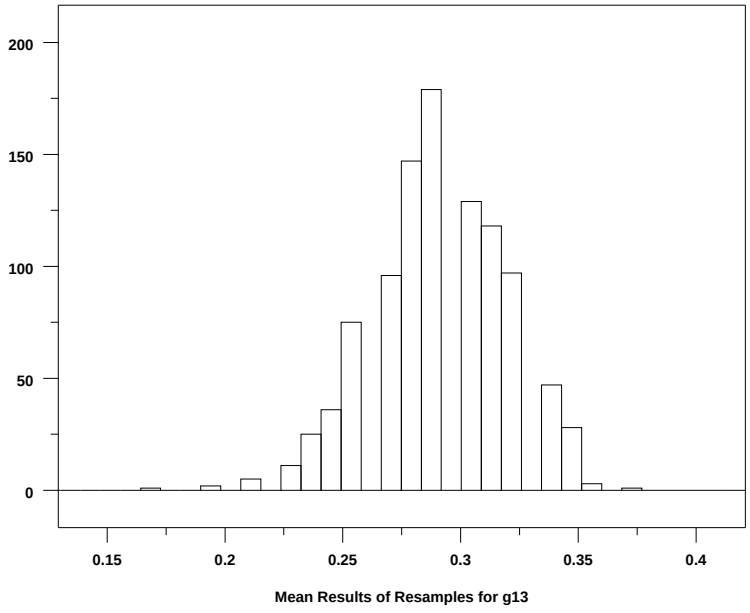


Figure C.8: Bootstrap distribution for the mean statistic for problem g13.

D

Proofs of Pareto Optimality for the Approach Based on a Vector of Goals

The following proofs correspond to Pareto optimality for the ε-constraint problem, found in [118]. Such proofs were adapted for its use with a vector of goals, instead of ε values. For such reason, we renamed the problem as the *goal-constraint problem*.

Definition 1. Let $\mathbf{z}^{\text{goal}} = (z_1^{\text{goal}}, \ldots, z_m^{\text{goal}})$ be a vector of goals. Thus, a *goal-constraint* problem is defined as follows:

$$\begin{array}{ll} \text{minimize} & f_i(\mathbf{x}) \\ \text{subject to} & f_j(\mathbf{x}) \leq z_j^{\text{goal}} \quad \forall j = \{1, \ldots, m\}, j \neq i \end{array}$$

This problem must be solved for all $i \in \{1, \ldots, m\}$, in order to obtain the vertices of the nearest region of the Pareto front to the vector of goals.

Theorem 1. The solution of a goal-constraint problem is weakly Pareto optimal.

Proof. Let $\mathbf{x}^\star$ be a solution of the goal-constraint problem. Assuming that $\mathbf{x}^\star$ is not weakly Pareto optimal, there exists another solution $\mathbf{x}^{\text{a}}$ such that $f_k(\mathbf{x}^{\text{a}}) < f_k(\mathbf{x}^\star)$ for all $k = 1, \ldots, m$. Such solution fulfills all the constraints of the goal-constraint problem, since $f_k(\mathbf{x}^{\text{a}}) < f_k(\mathbf{x}^\star) \leq z_k^{\text{goal}}$ for $k \neq i$. Moreover, for $k = i$, we have $f_i(\mathbf{x}^{\text{a}}) < f_i(\mathbf{x}^\star)$, which

contradicts the affirmation that $\mathbf{x}^\star$ is a solution of the goal-constraint problem. Thus, $\mathbf{x}^\star$ has to be weakly Pareto optimal.

Theorem 2. A solution $\mathbf{x}^\star$ of a goal-constraint problem is Pareto optimal if such a solution is unique for some i with $z_j^{\text{goal}} = f_j(\mathbf{x}^\star)$, $j \neq i$.

Proof. Let $\mathbf{x}^\star$ be a unique solution of the goal-constraint problem for some i. Assuming that $\mathbf{x}^\star$ is not Pareto optimal, there exists another solution $\mathbf{x}^{\text{a}}$ such that $f_k(\mathbf{x}^{\text{a}}) \leq f_k(\mathbf{x}^\star)$ for all $k = 1, \ldots, m$, and $f_\ell(\mathbf{x}^{\text{a}}) < f_\ell(\mathbf{x}^\star)$ for at least one ℓ. Because the solution $\mathbf{x}^\star$ is unique, it is the only minimum for all the feasible solutions $\mathbf{x}$ (the feasibility condition is $f_k(\mathbf{x}) \leq f_k(\mathbf{x}^\star)$, $k \neq i$), *i.e.*, $f_i(\mathbf{x}) > f_i(\mathbf{x}^\star)$. These relations contradict the previous ones. Thus, $\mathbf{x}^\star$ has to be Pareto optimal.

Theorem 3. The unique solution of a goal-constraint problem is Pareto optimal, regardless of the value of the vector of goals $\mathbf{z}^{\text{goal}} = (z_1^{\text{goal}}, \ldots, z_m^{\text{goal}})$.

Proof. Let $\mathbf{x}^\star$ be a unique solution of the goal-constraint problem. Assuming that $\mathbf{x}^\star$ is not Pareto optimal, there exists another solution $\mathbf{x}^{\text{a}}$ such that $f_k(\mathbf{x}^{\text{a}}) \leq f_k(\mathbf{x}^\star)$ for all $k = 1, \ldots, m$, and $f_\ell(\mathbf{x}^{\text{a}}) < f_\ell(\mathbf{x}^\star)$ for at least one ℓ. It may occur one of two cases:

- If $\ell = i$, then $f_i(\mathbf{x}^{\text{a}}) < f_i(\mathbf{x}^\star)$, which contradicts the affirmation that $\mathbf{x}^\star$ is a solution of the goal-constraint problem.
- If $\ell \neq i$, then $f_\ell(\mathbf{x}^{\text{a}}) < f_\ell(\mathbf{x}^\star) \leq z_\ell^{\text{goal}}$, $f_k(\mathbf{x}^{\text{a}}) \leq f_k(\mathbf{x}^\star) \leq z_k^{\text{goal}}$ for all $k \neq \ell$ and i, and $f_i(\mathbf{x}^{\text{a}}) \leq f_i(\mathbf{x}^\star)$. This last inequality contradicts the uniqueness of the solution.

Thus, $\mathbf{x}^\star$ has to be Pareto optimal.

Bibliography

[1] Hojjat Adeli and Nai-Tsang Cheng. Augmented Lagrangian Genetic Algorithm for Structural Optimization. *Journal of Aerospace Engineering*, 7(1):104–118, January 1994.

[2] Thomas Bäck, David Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*, volume 1. IOP Publishing Ltd. and Oxford University Press, 1997.

[3] James C. Bean. Genetics and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6(2):154–160, 1994.

[4] James C. Bean and Atidel Ben Hadj-Alouane. A Dual Genetic Algorithm for Bounded Integer Programs. Technical Report TR 92-53, Department of Industrial and Operations Engineering, The University of Michigan, 1992. To appear in R.A.I.R.O.-R.O. (invited submission to special issue on GAs and OR).

[5] Ashok Dhondu Belegundu. *A Study of Mathematical Programming Methods for Structural Optimization*. Department of civil and environmental engineering, University of Iowa, Iowa, Iowa, 1982.

[6] Sheela V. Belur. CORE: Constrained Optimization by Random Evolution. In John R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1997 Conference*, pages 280–286, Stanford University, California, July 1997. Stanford Bookstore.

[7] Jon Louis Bentley and Jerome H. Friedman. Data Structures for Range Searching. *ACM Computing Surveys*, 11(4):397–409, December 1979.

[8] George Bilchev and Ian C. Parmee. The Ant Colony Metaphor for Searching Continuous Design Spaces. In Terence C. Fogarty, editor, *Evolutionary Computing*, pages 25–39. Springer Verlag, Sheffield, UK, April 1995.

[9] George Bilchev and Ian C. Parmee. Constrained and Multi-Modal Optimisation with an Ant Colony Search Model. In Ian C. Parmee and M. J. Denham, editors, *Proceedings of 2nd International Conference on Adaptive Computing in Engineering Design and Control*. University of Plymouth, Plymouth, UK, March 1996.

[10] H. J. Bremermann. Optimization through evolution and recombination. In Marshall C. Yovitis and George T. Jacobi, editors, *Self-Organizing Systems*, pages 93–106. Spartan, Washington, D.C., 1962.

[11] Eduardo Camponogara and Sarosh N. Talukdar. A Genetic Algorithm for Constrained and Multiobjective Optimization. In Jarmo T. Alander, editor, *3rd Nordic Workshop on Genetic Algorithms and Their Applications (3NWGA)*, pages 49–62, Vaasa, Finland, August 1997. University of Vaasa.

[12] C. S. Chang, W. Wang, A. C. Liew, F. S. Wen, and D. Srinivasan. Genetic Algorithm Based Bicriterion Optimization for Traction Sustations in DC Railway System. In *Proceedings of the Second IEEE International Conference on Evolutionary Computation*, pages 11–16, Piscataway, New Jersey, 1995. IEEE Press.

[13] A. Charnes and W. W. Cooper. *Management Models and Industrial Applications of Linear Programming*, volume 1. John Wiley, New York, 1961.

[14] Chan-Jin Chung. *Knowledge-Based Approaches to Self-Adaptation in Cultural Algorithms*. PhD thesis, Wayne State University, Detroit, Michigan, 1997.

[15] Chan-Jin Chung and Robert G. Reynolds. A Testbed for Solving Optimization Problems using Cultural Algorithms. In Lawrence J. Fogel, Peter J. Angeline, and Thomas Bäck, editors, *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, Cambridge, Massachusetts, 1996. MIT Press.

[16] Chan-Jin Chung and Robert G. Reynolds. CAEP: An Evolution-based Tool for Real-Valued Function Optimization using Cultural Algorithms. *Journal on Artificial Intelligence Tools*, 7(3):239–292, 1998.

[17] Carlos A. Coello Coello and Nareli Cruz Cortés. A Parallel Implementation of an Artificial Immune System to Handle Constraints in Genetic Algorithms: Preliminary Results. In *Proceedings of the Congress on Evolutionary Computation 2002 (CEC'2002)*, volume 1, pages 819–824, Piscataway, New Jersey, May 2002. IEEE Service Center.

[18] Carlos A. Coello Coello. Constraint-handling using an evolutionary multiobjective optimization technique. *Civil Engineering and Environmental Systems*, 17:319–346, 2000.

[19] Carlos A. Coello Coello. Treating Constraints as Objectives for Single-Objective Evolutionary Optimization. *Engineering Optimization*, 32(3):275–308, 2000.

[20] Carlos A. Coello Coello. Use of a Self-Adaptive Penalty Approach for Engineering Optimization Problems. *Computers in Industry*, 41(2):113–127, January 2000.

[21] Carlos A. Coello Coello. Theoretical and Numerical Constraint-Handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art. *Computer Methods in Applied Mechanics and Engineering*, 191(11–12):1245–1287, January 2002.

[22] Carlos A. Coello Coello and Ricardo Landa Becerra. Adding knowledge and efficient data structures to evolutionary programming: A cultural algorithm for constrained optimization. In Erick Cantú-Paz et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2002)*, pages 201–209, San Francisco, California, July 2002. Morgan Kaufmann Publishers.

[23] Carlos A. Coello Coello and Ricardo Landa Becerra. Adding Knowledge and Efficient Data Structures to Evolutionary Programming: A Cultural Algorithm for Constrained Optimization. In W.B. Langdon, E.Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A.C.

Schultz, J. F. Miller, E. Burke, and N.Jonoska, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2002)*, pages 201–209, San Francisco, California, July 2002. Morgan Kaufmann Publishers.

[24] 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.

[25] 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.

[26] 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. ISBN 0-3064-6762-3.

[27] 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. ISBN 0-3064-6762-3.

[28] Carlos Artemio Coello Coello. *An Empirical Study of Evolutionary Techniques for Multiobjective Optimization in Engineering Design*. PhD thesis, Department of Computer Science, Tulane University, New Orleans, LA, April 1996.

[29] J. L. Cohon and D. H. Marks. A Review and Evaluation of Multiobjective Programming Techniques. *Water Resources Research*, 11(2):208–220, 1975.

[30] 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.

[31] 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, J. J. 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.

[32] Nareli Cruz-Cortés, Daniel Trejo-Pérez, and Carlos A. Coello Coello. Handling Constraints in Global Optimization using an Artificial Immune System. In Christian Jacob, Marcin L. Pilat, Peter J. Bentley, and Jonathan Timmis, editors, *Artificial Immune Systems. 4th International Conference, ICARIS 2005*, pages 234–247, Banff, Canada, August 2005. Springer. Lecture Notes in Computer Science Vol. 3627.

[33] Dragan Cvetković and Ian C. Parmee. Preferences and their Application in Evolutionary Multiobjective Optimisation. *IEEE Transactions on Evolutionary Computation*, 6(1):42–57, February 2002.

[34] George B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

[35] I. Das and J. Dennis. A Closer Look at Drawbacks of Minimizing Weighted Sums of Objectives for Pareto Set Generation in Multicriteria Optimization Problems. *Structural Optimization*, 14(1):63–69, 1997.

[36] Lawrence Davis, editor. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, New York, 1991.

[37] Kalyanmoy Deb. *Binary and Floating-Point Function Optimzation using Messy Genetic Algorithms*. PhD thesis, University of Alabama, Tuscaloosa, AL 35487, 1991. Department of Engineering Mechanics.

[38] Kalyanmoy Deb. An Efficient Constraint Handling Method for Genetic Algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2/4):311–338, 2000.

[39] Kalyanmoy Deb, Shamik Chaudhuri, and Kaisa Miettinen. Towards Estimating Nadir Objective Vector Using Evolutionary Approaches. In Maarten Keijzer et al., editor, *2006 Genetic and Evolutionary Computation Conference (GECCO'2006)*, volume 1, pages 643–650, Seattle, Washington, USA, July 2006. ACM Press. ISBN 1-59593-186-4.

[40] Kalyanmoy Deb and David E. Goldberg. An Investigation of Niche and Species Formation in Genetic Function Optimization. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 42–50, San Mateo, California, June 1989. George Mason University, Morgan Kaufmann Publishers.

[41] 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.

[42] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Trans. on Systems, Man, and Cybernetics–Part B*, 26(1):29–41, 1996.

[43] Leandro dos Santos Coelho and Viviana Cocco Mariani. An efficient particle swarm optimization approach based on cultural algorithm applied to mechanical design. In *Proceedings of the Congress on Evolutionary Computation (CEC 2006)*, pages 3844–3849. IEEE Service Center, 2006.

[44] W. H. Durham. *Co-evolution: Genes, Culture, and Human Diversity*. Stanford University Press, Stanford, California, 1994.

[45] 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.

[46] Larry J. Eshelman and J. Davis Schaffer. Real-coded Genetic Algorithms and Interval-Schemata. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 187–202. Morgan Kaufmann Publishers, San Mateo, California, 1993.

[47] A. Farhang-Mehr and S. Azarm. Minimal Sets of Quality Metrics. In Carlos M. Fonseca et al., editors, *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, pages 405–417, Faro, Portugal, April 2003. Springer. LNCS. Volume 2632.

[48] Eduardo Fernández and Juan Carlos Leyva. A method based on multiobjective optimization for deriving a ranking from a fuzzy preference relation. *European Journal of Operational Research*, 154(1):110–124, April 2004.

[49] David B. Fogel. An analysis of evolutionary programming. In David B. Fogel and Wirt Atmar, editors, *Proc. of the First Annual Conference on Evolutionary Programming*, pages 43–51, La Jolla, CA, 1992. Evolutionary Programming Society.

[50] Lawrence J. Fogel, editor. *Evolutionary Computation. The Fossil Record. Selected Readings on the History of Evolutionary Algorithms*. The Institute of Electrical and Electronic Engineers, New York, 1998.

[51] Lawrence J. Fogel, Alvin J. Owens, and Michael J. Walsh. *Artificial Intelligence Through Simulated Evolution*. John Wiley and Sons, New York, 1966.

[52] 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.

[53] 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.

[54] Benjamin Franklin and Marcel Bergerman. Cultural algorithms: Concepts and experiments. In *Proceedings of the 2000 Congress on Evolutionary Computation*, pages 1245–1251, Piscataway, New Jersey, 2000. IEEE Service Center.

[55] James M. Gere and William Weaver. *Analysis of Framed Structures*. D. Van Nostrand Company, Inc., 1965.

[56] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, New York, 1998.

[57] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[58] David E. Goldberg, Kalyanmoy Deb, Hillol Kargupta, and Georges Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 56–64, San Mateo, CA, 1993. Morgan Kaufman.

[59] Atidel Ben Hadj-Alouane and James C. Bean. A Genetic Algorithm for the Multiple-Choice Integer Program. *Operations Research*, 45:92–101, 1997.

[60] P. Hajela and J. Lee. Constrained Genetic Search via Schema Adaptation. An Immune Network Solution. In Niels Olhoff and George I. N. Rozvany, editors, *Proceedings of the First World Congress of Stuctural and Multidisciplinary Optimization*, pages 915–920, Goslar, Germany, 1995. Pergamon.

[61] P. Hajela and J. Lee. Constrained Genetic Search via Schema Adaptation. An Immune Network Solution. *Structural Optimization*, 12:11–15, 1996.

[62] P. Hajela and C. Y. Lin. Genetic search strategies in multicriterion optimal design. *Structural Optimization*, 4:99–107, 1992.

[63] P. Hajela and J. Yoo. Constraint Handling in Genetic Search Using Expression Strategies. *AIAA Journal*, 34(12):2414–2420, 1996.

[64] Sana Ben Hamida and Marc Schoenauer. ASCHEA: New Results Using Adaptive Segregational Constraint Handling. In *Proceedings of the Congress on Evolutionary Computation 2002 (CEC'2002)*, volume 1, pages 884–889, Piscataway, New Jersey, May 2002. IEEE Service Center.

[65] Robert Hinterding and Zbigniew Michalewicz. Your Brains and My Beauty: Parent Matching for Constrained Optimisation. In *Proceedings of the 5th International Conference on Evolutionary Computation*, pages 810–815, Anchorage, Alaska, May 1998.

[66] John H. Holland. Concerning efficient adaptive systems. In M. C. Yovitis, G. T. Jacobi, and G. D. Goldstein, editors, *Self-Organizing Systems*, pages 215–230. Spartan Books, Washington, D.C., 1962.

[67] John H. Holland. *Adaptation in Natural and Artifical Systems*. Ann Harbor, University of Michigan Press, 1975.

[68] Jeffrey Horn and Nicholas Nafpliotis. Multiobjective Optimization using the Niched Pareto Genetic Algorithm. Technical Report IlliGAl Report 93005, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, 1993.

[69] Xiaohui Hu, Russell C. Eberhart, and Yuhui Shi. Engineering Optimization with Particle Swarm. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, pages 53–57. Indianapolis, Indiana, USA, IEEE Service Center, April 2003.

[70] Simon Huband, Luigi Barone, Lyndon While, and Phil Hingston. A Scalable Multi-objective Test Problem Toolkit. In Carlos A. Coello Coello et al., editors, *Evolutionary Multi-Criterion Optimization. Third International Conference, EMO 2005*, pages 280–295, Guanajuato, México, March 2005. Springer. LNCS Vol. 3410.

[71] Simon Huband, Phil Hingston, Luigi Barone, and Lyndon While. A Review of Multiobjective Test Problems and a Scalable Test Problem Toolkit. *IEEE Transactions on Evolutionary Computation*, 10(5):477–506, October 2006.

[72] Radu Iacoban, Robert G. Reynolds, and Jon Brewster. Cultural Swarms: Assessing the Impact of Culture on Social Interaction and Problem Solving. In *2003 IEEE Swarm Intelligence Symposium Proceedings*, pages 212–219, Indianapolis, Indiana, USA, April 2003. IEEE Service Center.

[73] Radu Iacoban, Robert G. Reynolds, and Jon Brewster. Cultural Swarms: Modeling the Impact of Culture on Social Interaction and Problem Solving. In *2003 IEEE Swarm Intelligence Symposium Proceedings*, pages 205–211, Indianapolis, Indiana, USA, April 2003. IEEE Service Center.

[74] Y. Ijiri. *Management of Goals and Accounting for Control*. North-Holland, Amsterdan, 1965.

[75] Hisao Ishibuchi and Tadahiko Murata. Multi-Objective Genetic Local Search Algorithm and Its Application to Flowshop Scheduling. *IEEE Transactions on Systems, Man and Cybernetics*, 28(3):392–403, August 1998.

[76] 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.

[77] Andrzej Jaszkiewicz. Genetic local search for multiple objective combinatorial optimization. Technical Report RA-014/98, Institute of Computing Science, Poznan University of Technology, 1998.

[78] Fernando Jiménez and José L. Verdegay. Evolutionary techniques for constrained optimization problems. In Hans-Jürgen Zimmermann, editor, *7th European Congress on Intelligent Techniques and Soft Computing (EUFIT'99)*, Aachen, Germany, 1999. Verlag Mainz. ISBN 3-89653-808-X.

[79] Xidong Jin and Robert G. Reynolds. Using Knowledge-Based Evolutionary Computation to Solve Nonlinear Constraint Optimization Problems: a Cultural Algorithm Approach. In *1999 Congress on Evolutionary Computation*, pages 1672–1678, Washington, D.C., July 1999. IEEE Service Center.

[80] Xidong Jin and Robert G. Reynolds. Mining Knowledge in Large-Scale Databases Using Cultural Algorithms with Constraint Handling Mechanisms. In *Proceedings of the Congress on Evolutionary Computation 2000 (CEC'2000)*, volume 2, pages 1498–1506, Piscataway, New Jersey, July 2000. IEEE Service Center.

[81] H. Jutler. Liniejnaja modiel z nieskolkimi celevymi funkcjami (liner model with several objective functions). *Ekonomika i matematiceckije Metody*, 3:397–406, 1967.

[82] S. Kazarlis and V. Petridis. Varying Fitness Functions in Genetic Algorithms: Studying the Rate of Increase of the Dynamic Penalty Terms. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of the 5th Parallel Problem Solving from Nature (PPSN V)*, pages 211–220, Heidelberg, Germany, September 1998. Amsterdan, The Netherlands, Springer-Verlag. Lecture Notes in Computer Science Vol. 1498.

[83] James Kennedy and Russell C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco, California, 2001.

[84] J.-H. Kim and H. Myung. Evolutionary programming techniques for constrained optimization problems. *IEEE Transactions on Evolutionary Computation*, 1:129–140, July 1997.

[85] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[86] Hajime Kita, Yasuyuki Yabumoto, Naoki Mori, and Yoshikazu Nishikawa. Multi-Objective Optimization by Means of the Thermodynamical Genetic Algorithm. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature—PPSN IV*, Lecture Notes in Computer

Science, pages 504–512, Berlin, Germany, September 1996. Springer-Verlag.

[87] Joshua D. Knowles and David W. Corne. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[88] Ryszard Kowalczyk. Constraint Consistent Genetic Algorithms. In *Proceedings of the 1997 IEEE Conference on Evolutionary Computation*, pages 343–348, Indianapolis, USA, April 1997. IEEE.

[89] Slawomir Koziel and Zbigniew Michalewicz. A Decoder-based Evolutionary Algorithm for Constrained Parameter Optimization Problems. In T. Bäck, A. E. Eiben, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of the 5th Parallel Problem Solving from Nature (PPSN V)*, pages 231–240, Heidelberg, Germany, September 1998. Amsterdan, The Netherlands, Springer-Verlag. Lecture Notes in Computer Science Vol. 1498.

[90] Slawomir Koziel and Zbigniew Michalewicz. Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

[91] Frank Kursawe. A variant of evolution strategies for vector optimization. In H. P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature. 1st Workshop, PPSN I*, volume 496 of *Lecture Notes in Computer Science*, pages 193–197, Berlin, Germany, oct 1991. Springer-Verlag.

[92] Jouni Lampinen. Multi-Constrained Optimization by the Differential Evolution. In M.H. Hamza, editor, *Proceedings of the IASTED International Conference Artificial Intelligence and Applications (AIA 2001)*, pages 177–184, September 2001.

[93] Jouni Lampinen. Solving Problems Subject to Multiple Nonlinear Constraints by the Differential Evolution. In Radek Matousek & Pavel Osmera, editor, *Proceedings of MENDEL 2001, 7th International Conference on Soft Computing*, pages 50–57, June 2001.

[94] Jouni Lampinen. A Constraint Handling Approach for the Differential Evolution Algorithm. In *Proceedings of the Congress on Evolutionary Computation 2002 (CEC'2002)*, volume 2, pages 1468–1473, Piscataway, New Jersey, May 2002. IEEE Service Center.

[95] Ricardo Landa Becerra and Carlos A. Coello Coello. A Cultural Algorithm with Differential Evolution to Solve Constrained Optimization Problems. In Carlos A. Reyes Christian Lemaître and Jesús A. González, editors, *Advances in Artificial Intelligence – IBERAMIA 2004*, pages 881–890. Springer-Verlag, Lecture Notes in Artificial Intelligence Vol. 3315, November 2004.

[96] Ricardo Landa Becerra and Carlos A. Coello Coello. Culturizing Differential Evolution for Constrained Optimization. In Ricardo Baeza-Yates, J. Luis Marroquin, and Edgar Chávez, editors, *Proceedings of the Fifth International Conference on Computer Science (ENC 2004)*, pages 304–311. IEEE Computer Society, September 2004.

[97] Ricardo Landa Becerra and Carlos A. Coello Coello. Cultured Differential Evolution for Constrained Optimization. *Computer Methods in Applied Mechanics and Engineering*, 2005. in press.

[98] Ricardo Landa Becerra and Carlos A. Coello Coello. Optimization with Constraints using a Cultured Differential Evolution Approach. In Hans-Georg Beyer et al., editor, *Genetic and Evolutionary Computation Conference (GECCO'2005)*, volume 1, pages 27–34, Washington, DC, USA, June 2005. ACM Press. ISBN 1-59593-010-8.

[99] Ricardo Landa Becerra and Carlos A. Coello Coello. Solving hard multiobjective optimization problems using ε-constraint with cultured differential evolution. In Thomas Philip Runarsson, Hans-Georg Beyer, Edmund Burke, Juan J. Merelo-Guervós, L. Darrell Whitley, and Xin Yao, editors, *Parallel Problem Solving from Nature - PPSN IX, 9th International Conference*, pages 543–552. Springer. Lecture Notes in Computer Science Vol. 4193, Reykjavik, Iceland, September 2006.

[100] Ricardo Landa Becerra, Carlos A. Coello Coello, and Alfredo G. Hernandez-Diaz. Alternative techniques to solve hard multiobjective optimization problems. In *Proceedings of the Genetic*

and Evolutionary Computation Conference (GECCO'2007). ACM Press, 2007. (accepted).

[101] Marco Laumanns, Lothar Thiele, and Eckart Zitzler. An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. *European Journal of Operational Research*, 169:932–942, 2006.

[102] T. Van Le. A Fuzzy Evolutionary Approach to Constrained Optimization Problems. In *Proceedings of the Second IEEE Conference on Evolutionary Computation*, pages 274–278, Perth, November 1995. IEEE.

[103] G. E. Liepins and Michael D. Vose. Representational Issues in Genetic Optimization. *Journal of Experimental and Theoretical Computer Science*, 2(2):4–30, 1990.

[104] Gunar E. Liepins and W. D. Potter. A Genetic Algorithm Approach to Multiple-Fault Diagnosis. In Lawrence Davis, editor, *Handbook of Genetic Algorithms*, chapter 17, pages 237–250. Van Nostrand Reinhold, New York, New York, 1991.

[105] Yung-Chien Lin, Kao-Shing Hwang, and Feng-Sheng Wang. Hybrid Differential Evolution with Multiplier Updating Method for Nonlinear Constrained Optimization. In *Proceedings of the Congress on Evolutionary Computation 2002 (CEC'2002)*, volume 1, pages 872–877, Piscataway, New Jersey, May 2002. IEEE Service Center.

[106] Xiaojian Liu, D. W. Begg, and R. J. Fishwick. Genetic approach to optimal topology/controller design of adaptive structures. *International Journal for Numerical Methods in Engineering*, 41:815–830, 1998.

[107] Daniel H. Loughlin and S. Ranjithan. The Neighborhood constraint method: A Genetic Algorithm-Based Multiobjective Optimization Technique. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 666–673, San Mateo, California, July 1997. Michigan State University, Morgan Kaufmann Publishers.

[108] Sushil Louis and Gregory Rawlins. Designer genetic algorithms: Genetic algorithms in structure design. In Richard K. Belew and

Lashon B. Booker, editors, *Fourth International Conference on Genetic Algorithms*, pages 53–60, University of California, San Diego, jul 1991. Morgan Kauffman Publishers.

[109] Sushil J. Louis. *Genetic Algorithms as a Computational Tool for Design*. PhD thesis, Department of Computer Science, Indiana University, aug 1993.

[110] Sushil J. Louis and Fang Zhao. Domain knowledge for genetic algorithms. *International Journal of Expert Systems*, 8(3):195–211, 1995.

[111] Efrén Mezura-Montes and Carlos A. Coello Coello. Saving Evaluations in Differential Evolution for Constrained Optimization. In Vladimir Estivill-Castro and J. Alfredo Sánchez, editors, *Sixth Mexican International Conference on Computer Science (ENC'05)*, pages 274–281, Los Alamitos, California, September 2005. IEEE Computer Society Press.

[112] Efrén Mezura-Montes, Jesús Velázquez-Reyes, and Carlos A. Coello Coello. Promising Infeasibility and Multiple Offspring Incorporated to Differential Evolution for Constrained Optimization. In H.-G. Beyer, U.-M. O'Reilly, D.V. Arnold, W. Banzhaf, C. Blum, E.W. Bonabeau, E. Cantú Paz, D. Dasgupta, K. Deb, J.A. Foste r, E.D. de Jong, H. Lipson, X. Llora, S. Mancoridis, M. Pelikan, G.R. Raidl, T. Soule, A. Tyrrell, J.-P. Watson, and E. Zitzler, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2005)*, volume 1, pages 225–232, New York, June 2005. Washington DC, USA, ACM Press. ISBN 1-59593-010-8.

[113] Zbigniew Michalewicz and Naguib F. Attia. Evolutionary Optimization of Constrained Problems. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pages 98–108. World Scientific, 1994.

[114] Zbigniew Michalewicz and Cezary Z. Janikow. Handling Constraints in Genetic Algorithms. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 151–157, San Mateo, California, 1991. Morgan Kaufmann Publishers.

[115] Zbigniew Michalewicz and G. Nazhiyath. Genocop III: A co-evolutionary algorithm for numerical optimization with nonlinear constraints. In David B. Fogel, editor, *Proceedings of the Second IEEE International Conference on Evolutionary Computation*, pages 647–651, Piscataway, New Jersey, 1995. IEEE Press.

[116] Zbigniew Michalewicz and Marc Schoenauer. Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[117] Eric Michielssen, Jean-Michel Sajer, S. Ranjithan, and Raj Mittra. Design of Lightweight, Broad-Band Microwave Absorbers Using Genetic Algorithms. *IEEE Transactions on Microwave Theory and Techniques*, 41(6/7):1024–1031, 1993.

[118] K. Miettinen, M. M. Mäkelä, and J. Mäkinen. Handling Constraints with Penalty Techniques in Genetic Algorithms - A Numerical Comparison. Technical Report B10/1999, University of Jyväskylä, Department of Mathematical Information Technology, Series B, Scientific Computing, 1999.

[119] Tom Mitchell. *Version Spaces: An Approach to Concept Learning*. PhD thesis, Computer Science Department, Stanford University, Stanford, California, 1978.

[120] R. Morrison and K. De Jong. A test problem generator for non-stationary environments. In *Proceedings of the Congress on Evolutionary Computation (CEC 1999)*, pages 2047–2053. IEEE Service Center, 1999.

[121] Hyun Myung and Jong-Hwan Kim. Hybrid Interior-Lagrangian Penalty Based Evolutionary Optimization. In V.W. Porto, N. Saravanan, D. Waagen, and A.E. Eiben, editors, *Proceedings of the 7th International Conference on Evolutionary Programming (EP98)*, pages 85–94, Heidelberg, Germany, March 1998. San Diego, California, USA, Springer-Verlag. Lecture Notes in Computer Science Vol. 1447.

[122] J.A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[123] Angel E. Mu noz Zavala, Arturo Hernández-Aguirre, Enrique R. Villa-Diharce, and Salvador Botello-Rionda. PESO+ for Constrained Optimization. In *2006 IEEE Congress on Evolutionary Computation (CEC'2006)*, pages 935–942, Vancouver, BC, Canada, July 2006. IEEE.

[124] C. K. Oei, D. E. Goldberg, and S.-J. Chang. Tournament Selection, Niching, and the Preservation of Diversity. Technical Report Technical Report 91011, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, 1993.

[125] Tatsuya Okabe. *Evolutionary Multi-Objective Optimization - On the Distribution of Offspring in Parameter and Fitness Space* -. PhD thesis, Bielefeld University, Germany, 2004.

[126] Tatsuya Okabe, Yaochu Jin, Markus Olhofer, and Bernhard Sendhoff. On Test Functions for Evolutionary Multi-objective Optimization. In Xin Yao et al., editors, *Parallel Problem Solving from Nature - PPSN VIII*, pages 792–802, Birmingham, UK, September 2004. Springer-Verlag. LNCS Vol. 3242.

[127] A. Osyczka. *Multicriterion Optimization in Engineering with FORTRAN programs*. Ellis Horwood Limited, 1984.

[128] Andrzej Osyczka and Sourav Kundu. A new method to solve generalized multicriteria optimization problems using the simple genetic algorithm. *Structural Optimization*, 10:94–99, 1995.

[129] J. Paredis. Co-evolutionary Constraint Satisfaction. In *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature*, pages 46–55, New York, 1994. Springer Verlag.

[130] V. Pareto. *Cours D'Economie Politique*, volume I and II. F. Rouge, Lausanne, 1975.

[131] I. C. Parmee and G. Purchase. The development of a directed genetic search technique for heavily constrained design spaces. In I. C. Parmee, editor, *Adaptive Computing in Engineering Design and Control-'94*, pages 97–102, Plymouth, UK, 1994. University of Plymouth.

[132] Ian C. Parmee and Andrew H. Watson. Preliminary Airframe Design Using Co-Evolutionary Multiobjective Genetic Algorithms. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, volume 2, pages 1657–1665, San Francisco, California, July 1999. Morgan Kaufmann.

[133] K.E. Parsopoulos and M.N. Vrahatis. Particle Swarm Optimization Method for Constrained Optimization Problems. In P. Sincak, J.Vascak, V. Kvasnicka, and J. Pospicha, editors, *Intelligent Technologies - Theory and Applications: New Trends in Intelligent Technologies*, pages 214–220. IOS Press, 2002. Frontiers in Artificial Intelligence and Applications series, Vol. 76 ISBN: 1-58603-256-9.

[134] Bin Peng and Robert G. Reynolds. Cultural algorithms: Knowledge learning in dynamic environments. In *Proceedings of the Congress on Evolutionary Computation (CEC 2004)*, pages 1751–1758. IEEE Service Center, 2004.

[135] David Powell, Michael Skolnick, and S. Tong. EnGENEous : domain independent, machine learning for design implementation. In J. David Schaffer, editor, *Third International Conference on Genetic Algorithms*, pages 151–9, George Mason University, jun 1989. Morgan Kauffman Publishers.

[136] David Powell and Michael M. Skolnick. Using genetic algorithms in engineering design optimization with non-linear constraints. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 424–431. Morgan Kaufmann Publishers, jul 1993.

[137] David J. Powell, Michael M. Skolnick, and Siu Shing Tong. Interdigitation : Hybrid technique for engineering design optimization employing genetic algorithms, expert systems, and numerical optimization. In Lawrence Davis, editor, *Handbook of Genetic Algorithms*, chapter 20, pages 312–331. Van Nostrand Reinhold, New York, 1991.

[138] Kenneth V. Price. An introduction to differential evolution. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 79–108. McGraw-Hill, London, UK, 1999.

[139] Singiresu S. Rao. *Engineering Optimization*. John Wiley and Sons, third edition, 1996.

[140] Gregory J. Rawlins. Introduction. In Gregory J. Rawlins, editor, *Foundations of genetic algorithms*, pages 1–10. Morgan Kaufmann, San Mateo, CA, 1991.

[141] 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, July 2000. Morgan Kaufmann.

[142] Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzbog, Stuttgart, 1973. German.

[143] Colin B. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, Great Britain, 1993.

[144] B. Rekiek, P. de Lit, and A. Delchambre. Hybrid assembly line design and user's preferences. *International Journal of Production Research*, 40(5):1095–1111, March 2002.

[145] A. C. Renfrew. Dynamic Modeling in Archaeology: What, When, and Where? In S. E. van der Leeuw, editor, *Dynamical Modeling and the Study of Change in Archaelogy*. Edinburgh University Press, Edinburgh, Scotland, 1994.

[146] Robert G. Reynolds. An Introduction to Cultural Algorithms. In A. V. Sebald and L. J. Fogel, editors, *Proceedings of the Third Annual Conference on Evolutionary Programming*, pages 131–139. World Scientific, River Edge, New Jersey, 1994.

[147] Robert G. Reynolds. Cultural algorithms: Theory and applications. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 367–377. McGraw-Hill, London, 1999.

[148] Robert G. Reynolds, Zbigniew Michalewicz, and M. Cavaretta. Using cultural algorithms for constraint handling in GENOCOP. In

J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 298–305. MIT Press, Cambridge, Massachusetts, 1995.

[149] Jon T. Richardson, Mark R. Palmer, Gunar Liepins, and Mike Hilliard. Some Guidelines for Genetic Algorithms with Penalty Functions. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms (ICGA-89)*, pages 191–197, San Mateo, California, June 1989. George Mason University, Morgan Kaufmann Publishers.

[150] Rodolphe G. Le Riche, Catherine Knopf-Lenoir, and Raphael T. Haftka. A Segregated Genetic Algorithm for Constrained Structural Optimization. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA-95)*, pages 558–565, San Mateo, California, July 1995. University of Pittsburgh, Morgan Kaufmann Publishers.

[151] Peter J. Richerson and Robert Boyd. *Not by Genes Alone: How culture transformed human evolution*. The University of Chicago Press, 2005.

[152] Jon Rowe, Kevin Vinsen, and Nick Marvin. Parallel GAs for Multiobjective Functions. In Jarmo T. Alander, editor, *Proceedings of the Second Nordic Workshop on Genetic Algorithms and Their Applications (2NWGA)*, pages 61–70, Vaasa, Finland, August 1996. University of Vaasa.

[153] Thomas P. Runarsson and Xin Yao. Stochastic Ranking for Constrained Evolutionary Optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, September 2000.

[154] Saleh M. Saleem. *Knowledge-Based Solution to Dynamic Optimization Problems using Cultural Algorithms*. PhD thesis, Wayne State University, Detroit, Michigan, 2001.

[155] Eric Sandgren. Multicriteria design optimization by goal programming. In Hojjat Adeli, editor, *Advances in Design Optimization*, chapter 23, pages 225–265. Chapman & Hall, London, 1994.

[156] 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.

[157] Marc Schoenauer and Zbigniew Michalewicz. Evolutionary Computation at the Edge of Feasibility. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Proceedings of the Fourth Conference on Parallel Problem Solving from Nature (PPSN IV)*, pages 245–254, Heidelberg, Germany, September 1996. Berlin, Germany, Springer-Verlag.

[158] Marc Schoenauer and Spyros Xanthakis. Constrained GA Optimization. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA-93)*, pages 573–580, San Mateo, California, July 1993. University of Illinois at Urbana-Champaign, Morgan Kauffman Publishers.

[159] Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. Wiley, Chichester, 1981.

[160] David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC, Boca Raton, Florida, USA, 2004.

[161] R. Solich. Zadanie programowania liniowego z wieloma funkcjami celu (linear programming problem with several objective functions). *Przeglad Statystyczny*, 16:24–30, 1969.

[162] N. Srinivas and Kalyanmoy Deb. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3):221–248, Fall 1994.

[163] M. A. Stephens. Edf statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 69:730–737, 1974.

[164] Rainer Storn. On the Usage of Differential Evolution for Function Optimization. In *1996 Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS 1996)*, pages 519–523, Berkeley, 1996. IEEE.

[165] Rainer Storn. System Design by Constraint Adaptation and Differential Evolution. *IEEE Transactions on Evolutionary Computation*, 3(1):22–34, April 1999.

[166] Patrick D. Surry and Nicholas J. Radcliffe. The COMOGA Method: Constrained Optimisation by Multiobjective Genetic Algorithms. *Control and Cybernetics*, 26(3):391–412, 1997.

[167] Patrick D. Surry, Nicholas J. Radcliffe, and Ian D. Boyd. A Multi-Objective Approach to Constrained Optimisation of Gas Supply Networks : The COMOGA Method. In Terence C. Fogarty, editor, *Evolutionary Computing. AISB Workshop. Selected Papers*, Lecture Notes in Computer Science, pages 166–180. Springer-Verlag, Sheffield, U.K., 1995.

[168] Gilbert Syswerda. Schedule Optimization Using Genetic Algorithms. In Lawrence Davis, editor, *Handbook of Genetic Algorithms*, chapter 21, pages 332–349. Van Nostrand Reinhold, New York, New York, 1991.

[169] Gregorio Toscano-Pulido and Carlos A. Coello Coello. A Constraint-Handling Mechanism for Particle Swarm Optimization. In *Proceedings of the Congress on Evolutionary Computation 2004 (CEC'2004)*, volume 2, pages 1396–1403, Piscataway, New Jersey, June 2004. Portland, Oregon, USA, IEEE Service Center.

[170] C. H. Tseng and T. W. Lu. Minimax multiobjective optimization in structural design. *Int. J. Numerical Methods in Engineering*, 30:1213–1228, 1990.

[171] Effie Tsoi, Kit Po Wong, and Chun Che Fung. Hybrid GA/SA Algorithms for Evaluating Trade-off Between Economic Cost and Environmental Impact in Generation Dispatch. In David B. Fogel, editor, *Proceedings of teh Second IEEE Conference on Evolutionary Computation (ICEC'95)*, pages 132–137, Piscataway, New Jersey, 1995. IEEE Press.

[172] David A. Van Veldhuizen and Gary B. Lamont. Multiobjective Optimization with Messy Genetic Algorithms. In *Proceedings of the 2000 ACM Symposium on Applied Computing*, pages 470–476, Villa Olmo, Como, Italy, 2000. ACM.

[173] 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.

[174] P. B. Wienke, C. Lucasius, and G. Kateman. Multicriteria target optimization of analytical procedures using a genetic algorithm. *Analytica Chimica Acta*, 265(2):211–225, 1992.

[175] Stewart W. Wilson. Classifier systems and the animat problem. *Machine Learning*, 2(3):199–228, 1987.

[176] David H. Wolpert and William G. Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

[177] Xiaofeng Yang and Mitsuo Gen. Evolution program for bicriteria transportation problem. In M. Gen and T. Kobayashi, editors, *Proceedings of the 16th International Conference on Computers and Industrial Engineering*, pages 451–454, Ashikaga, Japan, 1994. Pergamon Press.

[178] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195, Summer 2000.

[179] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. In K. Giannakoglou, D. Tsahalis, J. Periaux, P. Papailou, and T. Fogarty, editors, *EUROGEN 2001. Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*, Athens, Greece, September 2001.

[180] 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.

[181] Jesse B. Zydallis, David A. Van Veldhuizen, and Gary B. Lamont. A Statistical Comparison of Multiobjective Evolutionary Algorithms Including the MOMGA–II. In Eckart Zitzler, Kalyanmoy

Deb, Lothar Thiele, Carlos A. Coello Coello, and David Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 226–240. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.