A Cellular Evolutionary Algorithm To Tackle Constrained Multiobjective Optimization Problems

By

Cosijopii García García

A dissertation submitted in partial fulfillment of the requirements for the degree of:

Master of Science in Computer Science

at

Instituto Nacional de Astrofísica, Óptica y Electrónica

October, 2020

Supervised by:

Dra. Alicia Morales Reyes

Dra. María Guadalupe Martínez Peñaloza

A Cellular Evolutionary Algorithm To Tackle Constrained Multiobjective Optimization Problems

Master's thesis

By:

Cosijopii Garcia Garcia

Supervised by:

Dra. Alicia Morales Reyes

Dra. María Guadalupe Martínez Peñaloza

Instituto Nacional de Astrofísica Óptica y Electrónica

Tonantzintla, Puebla. October 2020

Abstract

Parallel schemes in Evolutionary Algorithms (EAs) and, in particular, fine-grained or cellular Evolutionary Algorithms to tackle constrained multiobjective optimization problems have been barely investigated. Structural properties of cellular evolutionary algorithms balance exploration and exploitation stages during the search to avoid stagnation and possible premature convergence. Two new algorithms cMOGA/D (Cellular Multiobjective Genetic Algorithm based on Decomposition) and cMODE/D (Cellular Multiobjective Differential Evolution Algorithm based on Decomposition) in this thesis are proposed. Which take advantage of the structural properties of cellular evolutionary algorithms as well as Genetic Algorithms and Differential Evolution by combining them with well-established concepts of MOEA/D, such as, targeting multiobjective problems via multiple subproblems by Tchebycheff decomposition, its neighboring conceptual basis; and also combining a constrained multiobjective problem with Constraint Handling Technique (CHT) called push and pull search (PPS) and Improved Epsilon (IE). Three updating mechanisms were implemented: Synchronous, Line Sweep, and Asynchronous. These provide different behaviors when solutions are evolved throughout the mesh. Results obtained and validated through statistical analysis show that both proposals developed are competitive in terms of performance metrics and runtime. Moreover, in EAs based complexity analysis, cMOGA/D and cMODE/D are faster algorithms when comparing to MOEA/D-PPS.

Resumen

El uso de los esquemas paralelos en los algoritmos evolutivos (EA) y, en particular, los de grano fino o celulares para abordar problemas de optimización multiobjetivo restringidos han sido muy poco investigados. Las propiedades estructurales de los algoritmos evolutivos celulares equilibran las etapas de exploración y explotación durante la búsqueda para evitar el estancamiento y la posible convergencia prematura. Esta tesis propone dos nuevos algoritmos cMOGA/D (Cellular Multiobjective Genetic Algorithm based on Decomposition) y cMODE/D (Cellular Multiobjective Differential Evolution Algorithm based on Decomposition), que aprovechan las propiedades estructurales de los algoritmos evolutivos celulares, así como de los algoritmos genéticos y de la evolución diferencial, combinándolos con conceptos bien establecidos de MOEA/D, como, por ejemplo, la búsqueda de problemas multiobjetivos a través de múltiples subproblemas utilizando la descomposición de Tchebycheff, así como el concepto de vecindario; también combinando técnicas de manejo de restricciones como Push and Pull Search (PPS) e Improved Epsilon (IE). Se implementaron tres mecanismos de actualización: Síncrona, Line Sweep y Asíncrona. Estos proporcionan diferentes comportamientos cuando las soluciones se esparcen a través de la malla. Los resultados obtenidos y validados por medio del análisis estadístico muestran que ambas propuestas desarrolladas son competitivas en términos de métricas de rendimiento y tiempo de ejecución. Además, en el análisis empírico de complejidad para algoritmos evolutivos, cMOGA/D y cMODE/D son algoritmos más rápidos cuando se comparan con MOEA/D-PPS.

Acknowledgments

My sincere thanks to my thesis supervisors Dra. Alicia Morales Reyes and Dra. María Guadalupe Martínez Peñaloza, for all the effort that this thesis involved.

To INAOE for providing me with all the tools and facilities as well as my reviewers for helping me to improve this work.

Finally to CONACYT for scholarship No 719001 with which I was able to finish my studies.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

Evolutionary algorithms (EAs) have been widely used as optimization techniques to tackle multiobjective problems (MOPs) not only theoretical academic problems but also real-world problems within engineering and other application areas [1]. EAs are stochastic searching techniques that do not require derivatives and can deal with complex discontinuous and constrained landscapes. However, EAs do not guarantee to find the global optimal solution, instead they deliver a set of optimal solutions from which the user can determine an appropriate one according to his/her requirements. Thus, EAs are well-developed and tested and allow them to obtain accurate results.

EAs are population-based algorithmic techniques that rely on solutions' diversity to successfully target multiobjective optimization problems. Promoting diversity during the searching process is a very important task if the population's diversity is lost prematurely, tracking rich solutions regions throughout a problem's landscape becomes more difficult. Main evolutionary operations, recombination and mutation, implicitly balance exploration and exploitation stages during the search as well as induce diversity among solutions. Yet, EAs may suffer from premature convergence and therefore many mechanisms to promote and to balance diversity have been developed [1]. Decentralizing a population has been a natural way in EAs for maintaining and improving diversity. Decentralized EAs are classified by their population's grain in fine or cellular and coarse or distributed EAs. In decentralized populations either coarse or fine grain implicitly lead to a more loose evolutionary process in which a number of subpopulations evolve independently with predefined criteria for solutions exchange among them or a population topology (commonly a toroidally connected grid) induces a global exploration and a local exploitation trade-off. Decentralizing populations enable higher genetic differentiation and provide better search space sampling thus improving achieved solutions quality while maintaining computational cost equivalent to a standard EA [2].

1.1 Problem statement

In real-world applications, optimization problems usually have a considerable degree of complexity, and this complexity may be based on the presence of multiple-conflicting objectives and a number of constraints. Therefore, real-world optimization problems in areas such as design, distribution, and engineering are constrained multiobjective optimization problems (CMOPs) [3, 4, 5, 6, 7]. This has led to the development of multiple tools that try to solve them, from the classic mathematical models to the newest metaheuristics, in recent years, multiobjective evolutionary algorithms have had a good performance in this type of problems, since they achieve highly reliable solutions in a much shorter time. Therefore, evolutionary algorithmic techniques to tackle MOPs are constantly renewed, together with essential mechanisms to deal with constraints such as designing more capable constraint handling techniques (CHTs) and their interaction with canonical EAs components among others.

1.2 Justification

Using parallel schemes has improved the performance of bio-inspired algorithms particularly evolutionary ones and good results have also been obtained when tackling single and multiobjective optimization problems(MOPs) [8]. For example, a combination of the cellular model with a multiobjective evolutionary algorithm (MOEA) achieves good convergence and improve metrics such as hypervolume [9]. On the other hand, swarm-based algorithms have also been combined with a cellular model making these behave with very good results [10]. Using islands and hybrid models have also shown better results than non-decentralized base algorithms [11, 8].

The proposed research aims at developing a parallel MOEA to tackle constrained multi-objective optimization problems (CMOPs). In general, MOPs have been targeted while taking advantage of their independence of solutions via decentralized algorithmic schemes. However, there is a lack of research in terms of targeting constrained multiobjective optimization problems through parallel schemes by decentralizing the population in fine or coarse grain subpopulations.

1.3 Hypothesis

Decentralized population schemes in multiobjective evolutionary algorithms help improve the performance of most recent approaches and reduce their computational cost.

1.4 General objective

To design, to develop, and to evaluate a parallel multiobjective evolutionary algorithm that handles constraints to achieve competitive performance in comparison to the state of the art solutions.

1.4.1 Specific objectives

- To study the state of the art regarding constrained MOEAs and parallel schemes to analyze their mechanisms, advantages, and disadvantages. The aim is to identify possible improvements that lead to the design of a new parallel constrained MOEA.

- To advance the knowledge within parallel constrained multiobjective optimization by developing an efficient constrained multiobjective parallel evolutionary algorithm which combines the advantages of constrained MOEAs and parallel approaches. This algorithm must comply with desirable features (efficiency, fast convergence and diversity control).

- To define the test functions that will be used to measure the performance of the algorithm that will be developed.

- To evaluate and validate the performance of the proposed algorithm with respect to other popular state-of-the-art constrained MOEAs. This validation will be done with a set of test problems and performance metrics representative of the multiobjective optimization community.

- To understand and analyze the role that the parallel mechanism plays in the performance of the proposed approach.

1.5 Methodology

The proposed methodology is shown in Figure 1.1. As a first step, the state of the art regarding constrained MOEAs and parallel constrained MOEAs will be studied. After that, a study of different parallel schemes and constraint handling techniques (CHTs) will be conducted. Having all the evidence on the performance of those two areas, a new algorithmic approach to tackle constrained multiobjetive problems considering parallel population dynamics is developed. To evaluate the performance of the proposed approach, some popular state of the art algorithms will be chosen to perform a comparative study. In addition, classical test problems and different performance metrics will be chosen to measure the performance of the new parallel constrained MOEA.

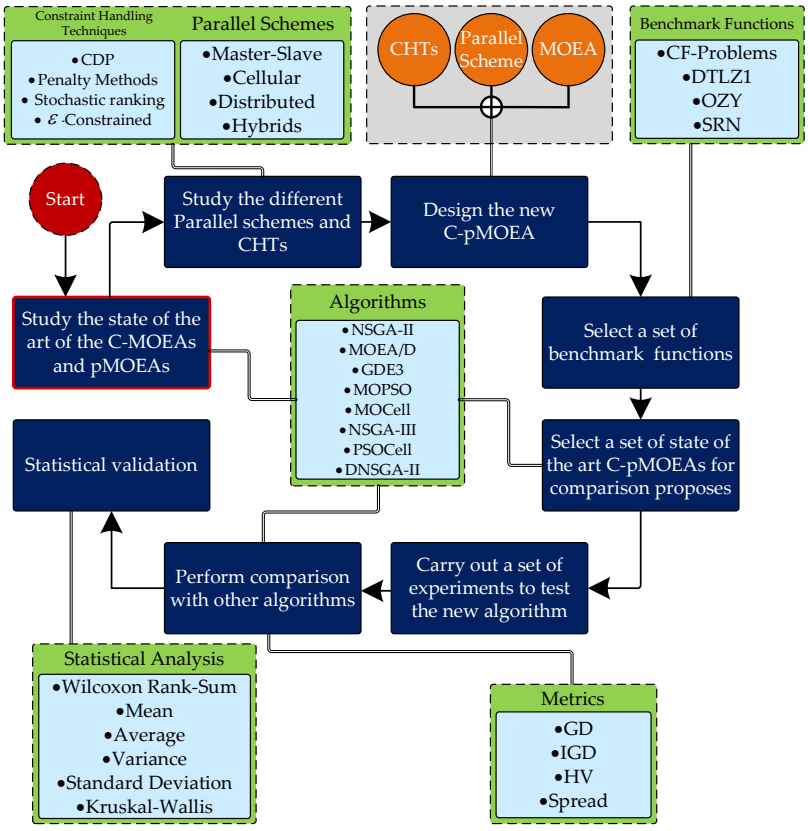


Figure 1.1: Proposed methodology.

1.6 Expected outcomes

- A new parallel constrained MOEA which combines different mechanisms to deal with constraints and has better performance than most recent approaches in the literature.
- A detailed empirical study of the proposed approach. This study is thoroughly validated by statistical testing and analysis considering standard parallel constrained MOEAs and standard benchmark problems.

1.6.1 Publications

Cosijopii Garcia-Garcia, María-Guadalupe Martínez-Peñaloza, Alicia Morales-Reyes. 2020. cMOGA/D: a novel cellular GA based on decomposition to tackle constrained multiobjetive problems. In Genetic and Evolutionary Computation Conference Companion (GECCO '20 Companion), July 8,12,2020, Cancún, Mexico. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3377929.3398137

1.7 Thesis organization

This thesis document is organized in 6 chapters. The first three (including this one) describe the basic concepts required to make this document as self-contained as possible.

Chapter 2 describes the background of the multiobjective optimization problems as well as an analysis of the different MOEAs operators, to finish with an analysis of the parallel schemes applied to evolutionary algorithms and the assessment metrics for multiobjective optimization. In chapter 3 the state of the art regarding MOEAs as well as parallel MOEAs and parallel constrained MOEAs are described. Chapters 4 and 5, focus on explaining in detail the two proposed algorithms cMOGA/D and cMODE/D. Finally, in Chapter 6 the general conclusions, as well as future work, are given.

Chapter 2
Background

This chapter provides an overview of the main topics involved in this research. Algorithmic techniques from the Evolutionary Computation area considering its main operations. After, an introduction to multiobjective optimization together with principal associated multiobjective evolutionary algorithms are described. Parallel approaches of evolutionary algorithms will be explained as well as constraint handling techniques. Finally, a detailed description of the mainly used performance metrics in the area is also included.

2.1 Evolutionary algorithms

Evolutionary algorithms (EAs) are a subset of algorithmic techniques within the Evolutionary Computation (EC) area which has been roughly considered as a scope within Artificial Intelligence (AI), and more accurately as a Soft Computing field of study. EAs are inspired in the process of natural evolution as they try to simulate the main behavior of Darwin's theory of evolution, and are commonly applied in search and optimization processes [12]. There are different application contexts for evolutionary algorithms, but the main ones are focused on optimization, in this area, three different kinds of problems can be tackled: single, multi, and many objectives optimization.

Most evolutionary algorithms share the same general stages, they fall in the category of generate and test algorithms, where the search is guided by stochastic variations through different operators [12]. Evolutionary algorithms are based on a set of solutions or a population, these solutions interact through recombination and mutation (main genetic operations), by mixing information from two or more candidates to create a new one and inducing small changes within solutions and thus introducing diversity. In Figure 2.1 a general scheme of an evolutionary algorithm is drawn.

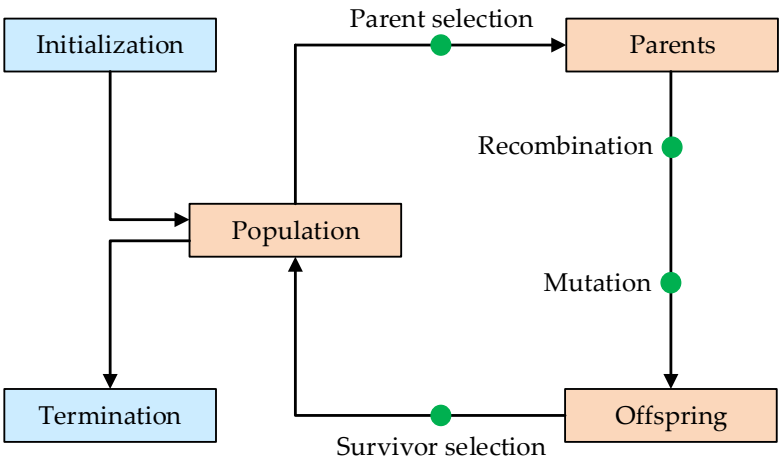


Figure 2.1: General scheme of Evolutionary algorithms.

In the following subsections, principal components and stages (selection, recombination, mutation, and replacement) of evolutionary algorithms will be explained. Then, two principal techniques related to this thesis will be detailed.

2.1.1 Solutions representation

The first step in the design of some evolutionary algorithms is to decide the representation that will be used by the candidates for solutions, this involves two terms, the genotype and its mapping to the phenotype. There are different ways to represent the problems, depending on the type of representation that is used the evolutionary operators, the mutation and recombination or crossover can vary [12, 13]. The following is a review of the different types of representations most commonly used.

Binary representation

This representation was the first to be used, and historically many genetic algorithms (GA), used this representation regardless of the context of the problem to be solved, this representation is based on strings of bits that represent the genotype (010101 . . . 1). In this type of representation, it is defined how long the bit string will be, and how it will be mapped to the phenotype, this depends on the type of problem, the bit strings can easily be used to represent real and integer values. For some problems where Boolean decision variables are needed, genotype-phenotype mapping is done naturally, an example of this

is the famous knapsack problem [12, 14]. An alternative coding that has been used in some GAs has been the Gray coding, in this coding, it is not required multiple changes in the bits to generate an integer value, this means that close solutions in the space of integer search, requires fewer operations to guide the search since this coding the strings of bits always have as a distance of Hamming one [12, 13]. For the binary representation, multiple recombination schemes are proposed, these start with the use of two parents to generate two offspring as the One-Point Crossover, n-Point Crossover, and Uniform Crossover. There are schemes to be used with multiple parents [15], also there are situations where only one offspring is considered, on the other hand, for the mutation operator, the most used scheme is to perform a bitwise in the string-bit with a certain probability Pm [12], in Figure 2.2 it can be seen an example of One-Point crossover and Bitwise mutation.

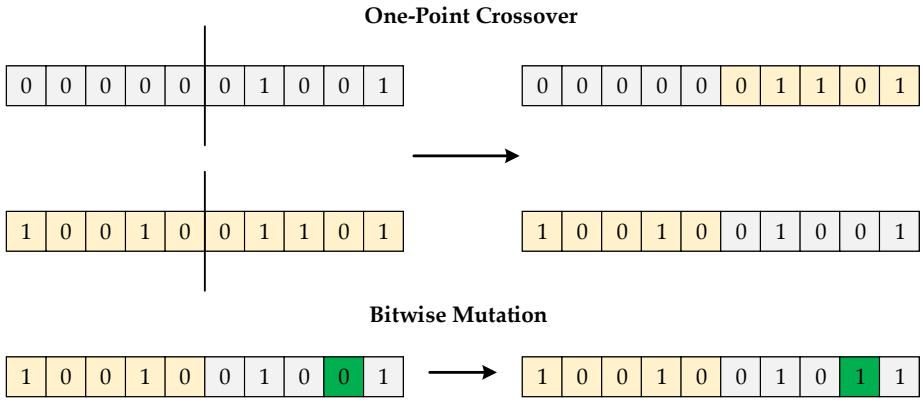


Figure 2.2: Example of One-Point Crossover and Bitwise Mutation.

Integer representation

In this type of representation, it is mostly used when it is required to find optimal values for a set of variables in the domain of integer values. These values can be unrestricted or restricted to some finite set. For example, if we try to find a route in a square grid, and we are restricted to the set $\{0, 1, 2, 3\}$ which represents { *North, East, South, West* }, in this case using an integer coding, is better than a binary, another example of this can be the representation of networks. For the recombination of this type of representation, it is normal to use the same set of operations that the binary representation, since when applying these operations the solutions are not outside the space of genotypes. Otherwise,

there are two main types of forms exist to make the mutation in integer representations the first one is called Random Resetting, where a change of a gene of the string of integers, by some number of random of a set of permissible values is made, this is made with a Pm probability, on the other hand, the second scheme of mutation is called Creep Mutation, in this scheme small values are added (positive or negative) to the gene that will be mutated, this with a p probability, these values are generated randomly and it is designed for ordinal attributes [12].

Real-valued representation

For many problems, genotype representation with real-valued is the most natural way of representation and current optimization applications use real-valued coding [13]. This occurs when variables values come from a continuous distribution rather than a discrete distribution. An example of this consists of physical quantities representation as length, width, height, or weight, some of these components can be real number values. Another example can be using an EA to evolve weights of connections of an artificial neural network. Considering limitations of precision that computers have, real values can be limited, therefore it is better to refer to them as floating-point numbers [12]. This genotype representation with k genes is a vector $\langle x_1, \ldots, x_k \rangle$ with $x_i \in \mathbb{R}$. In the literature, first evolutionary algorithms that dealt with this type of representation naturally were evolutionary strategies and evolutionary programming, which operated directly on real-valued vectors, although later genetic algorithms adapted recombination and mutation operators to this domain [14]. The following sections will give a more in-depth explanation of the crossing and mutation methods for real-valued representations.

Other representations

Different from representations already described above, there is a diversity of ways to represent a genotype. These types of representations, as well as those described above, are linked to the problem or area where the algorithm is developed. Permutation representation is given when there is a fixed set of values, which according to an order in which they are arranged represent an event, an example of this is the famous TSP (traveling salesman problem) in which vector's order represents the cities to travel thus the route to take. Normally, this set of values is represented with integers values, unlike other representations where numbers can be repeated in genotypes here is not allowed. This leads to a greater

complexity when performing operations of mutation and crossover, thus new methods have been developed such as PMX (Partially Mapped Crossover), Edge Crossover, Order Crossover, and Cycle crossover. On the other hand for mutation, Swap Mutation, Insert Mutation, and Scramble Mutation methods have been created [12, 14].

There are other types of representations such as mixed, where integer values are combined with real values. They are used in fuzzy logic systems or neural networks to optimize parameters that combine different types of values. [14]. Another not so traditional approach is using Introns, this approach is based on non-coding regions in vectors, and a last approach is Diploid representation which can include multiple values in alleles for one position in the genome [14].

2.1.2 Selection methods

Selection is one of the main operators used in evolutionary algorithms. Its main objective is to find the best solutions in a population. Identifying good or bad candidates depends on quality of solutions concerning their fitness or other value that describes their quality [14]. The essential idea is that if a solution has good fitness it is more likely to be selected. Selection criteria determine selection pressure, which is the degree to which good fitness solutions are selected. If selection pressure is too low, information from good parents will be spread too slowly throughout the population, and this is a very inefficient process. If selection pressure is too high, population will be stuck in a local optima. Good selection strategies allow exploitation of high fitness individuals in the population without losing diversity too quickly [13]. According to [16, 13], selection techniques can be classified in two categories **Fitness proportionate** and **Ordinal selection**.

Fitness proportionate

it includes methods such as roulette-wheel selection and universal stochastic selection. Each individual in the population is assigned a slot in the roulette wheel. This value is proportional to the fitness (or other measures that estimates how good a solution is for cases with multiple objectives). Therefore, in the roulette-wheel method good solutions have larger slots than bad solution. This means that good solutions most of the time will be selected and solutions with low fitness will be very little or not selected at all [16, 12].

Ordinal selection

Ordinal selection includes methods such as tournament selection and truncation selection. In tournament selection, p solutions are selected and enter into a tournament against each other. An individual with higher fitness in a group of p solutions wins the tournament and is selected as a parent. The most used tournament size is $p = 2$. Using this scheme, n tournaments are required for n individuals. In truncation selection, individuals are ordered according to their fitness value and top $(1/p)$ best ones are chosen to perform recombination [16, 12].

2.1.3 Recombination for real-valued representation

Recombination is the process by which a new solution is created using information from two or more parents. It is considered one of the most important characteristics in evolutionary algorithms. A traditional way in which recombination operators perform "crossover" is by marking sub-segments in parents genomes to later assemble into a new individual [12]. An example of this is Figure 2.2 where single-point crossover is illustrated. The term recombination is more general, usually, crossover term is used more often, this is motivated by the biological analogy of meiosis [14]. Recombination can be applied in various ways, many of these are derived from studies conducted in the community of evolutionary strategies as these are more related to continuous optimization problems. Taking as an analogy recombination operator for binary solutions, in which combinations of existing values are made to create offspring is called *discrete recombination*. In addition, there are other types like *arithmetic recombination* or *blend recombination*. All of them try to deal with new genetic material creation from selected parents [12]. Currently, *Simulated Binary Crossover* [17] operator is widely used for parent's recombination in continuous domain problems, as it has proven to have superior performance than other types of recombination [17]. In addition to being widely used in multiobjective domain [18, 19, 9, 20, 21, 22, 23]. The following is a review of different methods for real-valued recombination.

Simple arithmetic recombination

In simple arithmetic recombination, a point k is chosen. Then, for offspring 1, the first k float numbers in the solution are taken from parent 1 and placed in an offspring, while the

rest of the offspring is calculated by using arithmetic average from parent 1 and 2 [12].

$$\text{Offspring 1: } \langle x_1, \ldots, x_k, \alpha \cdot y_{k+1} + (1-\alpha) \cdot x_{k+1}, \ldots, \alpha \cdot y_n + (1-\alpha) \cdot x_n \rangle \text{ with } \alpha \in [0,1] \quad (2.1.1)$$

For offspring 2, procedure occurs with x and y reversed. An example is show in Figure 2.3.

Single arithmetic recombination

In single arithmetic recombination a random k allele is chosen. At its position, arithmetic average of the two parents is calculated ($\alpha \cdot y_k + (1-\alpha) \cdot x_k$), the other alleles are not modified. [12].

$$\text{Offspring 1: } \langle x_1, \ldots, x_{k-1}, \alpha \cdot y_k + (1-\alpha) \cdot x_k, x_{k+1}, \ldots, x_n \rangle \text{ with } \alpha \in [0,1] \quad (2.1.2)$$

For offspring 2, procedure happens with x and y reversed. An example is show in Figure 2.3.

Whole arithmetic recombination

Whole arithmetic recombination averages both parents, using the following expressions for each allele i in the offspring [14, 12].

$$\text{Offspring 1: } \alpha \cdot x_i + (1-\alpha) \cdot y_i, \text{ Offspring 2: } \alpha \cdot y_i + (1-\alpha) \cdot x_i \quad (2.1.3)$$

An example is show in Figure 2.3.

Blend Crossover

Blend Crossover (BLX-α) was introduced as a way to create offspring in regions larger than those generated by individual parents. This extra space is proportional to the distance between the parents [12]. To create descendants, a random number u is chosen from $[0, 1]$ and then calculate $\gamma = (1-2\alpha)u - \alpha$ and set in:

$$z_i = (1-\gamma)x_i + \gamma y_i \quad (2.1.4)$$

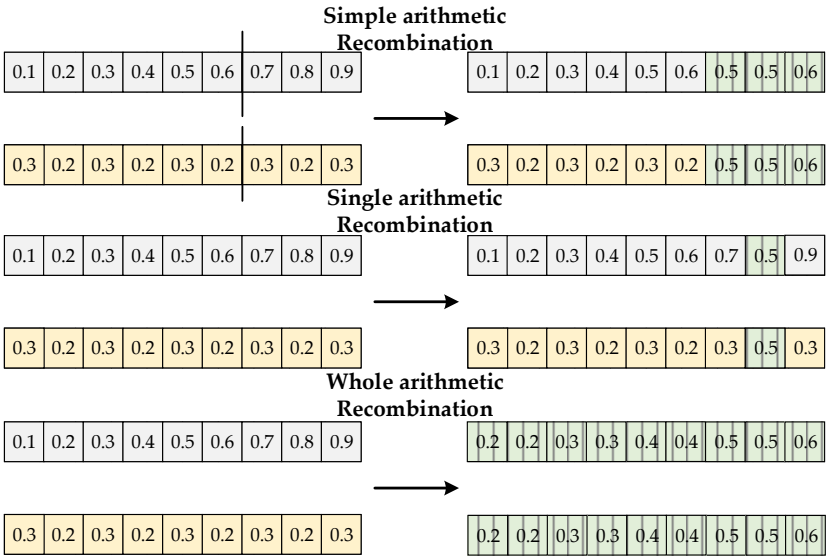


Figure 2.3: Simple, Single and Whole arithmetic recombination example, with $\alpha = \frac{1}{2}$.

Authors report that the best results are obtained with $\alpha = 0.5$. This gives the same probability for generating values within and outside ranges established by parents. Thus a balance between exploitation and exploration is obtained [12].

Simulated Binary Crossover

Simulated Binary Crossover (SBX) operator as its name implies tries to simulate single point binary recombination for real numbers solutions representation. This operator was created by Deb and Agrawal [17] and currently is the most used both in single objective and multiobjective problems. SBX works as follows:

1.- Choose a random number $u_i \in [0, 1]$

2.- β_{qi} is calculated, using :

$$\beta_{qi} = \begin{cases} (2u_i)^{\frac{1}{\eta+1}} & \text{if } u_i < 0.5 \\ \left(\frac{1}{2(1-u_i)}\right)^{\frac{1}{\eta+1}} & \text{otherwise} \end{cases} \tag{2.1.5}$$

where η is the distribution index which should be a non-negative number. Large η values increase the probability of creating near-parent solutions and small values allow to select distant values to generate the offspring.

3.- Finally, offspring are calculated using the following equations:

$$x_i^{(1,t+1)} = 0.5 \left[\left(1 + \beta_{qi}\right) x_i^{(1,t)} + \left(1 - \beta_{qi}\right) x_i^{(2,t)} \right] \tag{2.1.6}$$

$$x_i^{(2,t+1)} = 0.5 \left[\left(1 + \beta_{qi}\right) x_i^{(1,t)} + \left(1 - \beta_{qi}\right) x_i^{(2,t)} \right] \tag{2.1.7}$$

SBX has two main properties. First, the difference between the offspring is proportion to parents solutions and second, solutions close to their parents are monotonically more likely to be chosen as offspring than solutions located far from their parents. Therefore, parents who share genes are more likely to inherit these to the children [17, 24, 25].

2.1.4 Mutation for real-valued representation

Mutation is a mechanism in which only one solution is involved. Solutions selected as parents or offspring solutions are mutated by slightly modifying their genetic information [26, 14]. About For real-valued representation, where each individual in a population is a n-dimensional vector $x \in \mathbb{R}$. There are different methods to create new descendants using the mutation operator or to modify those recombined solutions. The most common way is to change the value of an allele for a random value generated within a lower bound L_i and an upper bound U_i. There are many ways to perform mutation in continuous domains, from simple approaches such as uniform or nonuniform mutation to self-adaptive methods [12], but more recently the use of polynomial mutation [24] has been widely used in single and multiobjective optimization and most often when combined with SBX operator which has overcome other recombination operators to this day. [18, 19, 9, 20, 21, 22, 23, 27, 28, 29].

Uniform mutation

For this operator x_i values are shifted by uniformly generated random values of $[L_i, U_i]$. This is the simplest mutation approach, analogous to bit wise mutation in binary encoding and random resetting for integer representations [12].

Nonuniform mutation

Nonuniform mutation introduces small changes in a solution's vector. This changed value is randomly generated form a Gaussian distribution, see Equation 2.1.8. In this equation

probability for generating a random number with any given magnitude tends to decrease rapidly as a function of the standard deviation σ. Changes tend to be small in most cases, but there is a probability that large changes are generated. This probability is defined with a σ parameter that determines which x_i values will be disturbed by the mutation operator. This criterion is commonly called *mutation step size*. An alternative to the Gaussian distribution is to use a Cauchy distribution since the probability of generating large values is slightly higher while using the same standard deviation [12].

$$p(\Delta x_i) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(\Delta x_i - \xi)^2}{2\sigma^2}} \tag{2.1.8}$$

Polynomial mutation

Similar to SBX recombination operator, in polynomial mutation the probability distribution can be a polynomial function instead of a normal distribution. In [24, 30], it was concluded that (η_m) parameter induces a $O\left((b-a)/\eta_m\right)$ perturbation effect in the solution. a and b are upper and lower limits of the problem, where $\eta_m \in [20, 100]$ is a common value or default value for η_m is when $\eta_m = 20$. For a p solution, p' is a mutated solution created as follows:

1.- First, choose a random number $u \in [0, 1]$.

2.- Finally, the following equation applies:

$$p' = \begin{cases} p + \overline{\delta_L}\left(P - x_i^{(L)}\right) & \text{for } u \leq 0.5 \\ p + \overline{\delta_R}\left(x_i^{(U)} - P\right) & \text{for } u > 0.5 \end{cases} \tag{2.1.9}$$

where $\overline{\delta_L}$ and $\overline{\delta_R}$ are calculated as follow:

$$\overline{\delta_L} = (2u)^{\frac{1}{(1+\eta_m)}} - 1, \text{for } u \leq 0.5 \tag{2.1.10}$$

$$\overline{\delta_R} = 1 - (2(1-u))\frac{1}{1+\eta_m}, \text{for } u > 0.5 \tag{2.1.11}$$

2.1.5 Replacement criterion

Survivors selection mechanism is responsible for managing a reduction process in a EA's population from a set of μ parents and λ offspring to a set of μ individuals that form the next generation. There are many ways to do this, but a main one is to decide based on individuals fitness. Another technique is based on population's age, this method is used in the simple genetic algorithm, where each individual exists only one cycle and parents are discarded to be replaced by offspring. This criterion is known as a *generational population model* [12, 16]. There are a number of fitness based strategies, considering μ parents $+\lambda$ offspring that will be passed on to the next generation, these are explained next.

Replace the worst

In this replacement scheme μ worst parents are replaced. This can lead to a premature convergence, when solutions get stuck in a limited search space zone. Thus diversity is reduced among solutions and therefore no major changes occur in the population. However, it is possible that poor fitness solutions can improve solutions quality by providing genetic diversity to the population [12, 16].

Elitism

Elitism is used to maintain the best solution(s) in the population. Thus if the best solution is chosen to be replaced, and any offspring is worse or equal. The offspring is discarded and the best individual is kept. This scheme can be combined with the approach of replacing the worst solution [16, 12].

$(\mu + \lambda)$ Selection

$(\mu + \lambda)$ selection comes from evolutionary strategies. It refers to the case in which both sets of parents and offspring are ranked in a same set. Then top μ solutions are kept for the next generation. This strategy can be seen as a generalization of replacing the worst criterion [12].

2.2 Genetic algorithms

The genetic algorithm (GA) is one of the most popular evolutionary algorithms. Its research methodology is based on natural evolution, and was initially conceived by Holland as a way to study adaptive behavior. The canonical or simple genetic algorithm, has a binary representation, fitness proportionate selection, and low mutation probability. [16, 12]. Over the years, new characteristics were developed, one of the most important is elitism, as well as different types of recombination and mutation operators [12]. According to [16] the genetic algorithm follows six stages, which are listed below:

1.- **Initialization:** The initial population of solutions is randomly generated across the search space, as well as it can be generated using knowledge of the problem domain or with previously gathered information.

2.- **Evaluation:** Once the population is created, the fitness value of every solution in the population is calculated.

3.- **Selection:** At the selection stage solutions chosen according to their fitness value. There are several forms of selection procedures, among them are roulette-wheel selection, stochastic selection, ranking selection, and tournament selection.

4.- **Recombination:** Information from two or more parents are combined to create a new possible better solution. There are many ways to achieve this depending on solutions representation.

5.- **Mutation:** Locally and randomly modifies a solution. It involves one or more ways of adding small perturbations to an individual.

6.- **Replacement:** Offspring created by selection, recombination, and mutation replaces the original population by some criterion. Among replacement techniques are elitist replacement, generation-wise replacement, and steady-state replacement.

These steps are repeated (2-6) until a stopping criterion is reached. It can be until a maximum number of evaluations, generations, or the desired solution is reached. In algorithm 1 a pseudo-code of a canonical or simple GA is shown.

Algorithm 1: Simple or Canonical GA

```
1  t = 0;
2  Initialize Population(t);
3  Evaluate Population(t);
4  while Termination condition not satisfied do
5      t = t + 1;
6      Select m(t) Parents from Population(t − 1);
7      Recombine and mutate solutions in m(t);
8      Create offspring population m′(t);
9      Evaluate m′(t);
10     Select individuals for next generation;
```

2.3 Differential evolution

The differential evolution (DE) algorithm is one of the most popular evolutionary algorithms. It was proposed in 1997 [31, 12]. Since DE will be the basis for one of the algorithmic proposals, it will be explained in detail. DE differs from GA in the reproduction mechanism and shares most stages of evolutionary algorithms. DE uses information of a current population to guide the search process, it also applies mutation operator first, different to other EAs [32]. There are different DE variants,these are classified with *DE/a/b/c* notation; *a* represents selection type for the base vector, *b* is the number of different vectors used and *c* defines the mutation scheme. The canonical version of differential evolution is *DE/rand/1/bin* where the base vector selection is random, only one vector is occupied as a mutant vector and crossover scheme is binomial although this also has its exponential variant *DE/rand/1/exp*.

There are other variants such as *DE/best/1/bin*. In this scheme, the base vector is a solution with the highest fitness in a current population. On the other hand, *DE/rand/2/bin* variant, difference operation is performed twice over 2 pairs of vectors with the idea of increasing diversity of the generated trial vector [13, 12].

DE follows the same steps in the evolutionary process explained in diagram 2.1, with a difference on mutation which is applied before recombination. It also adds a new operator, called differential mutation [31, 12]. In this work, differential operators offered by DE are used. They have been widely used in multiobjective optimization in combination with MOEA/D algorithm (section 2.5.2) [19, 28, 27].

At first, a candidate population $x_{i,G}, i = 1, 2, \ldots, NP$ is created, where $x_{i,G}$ is the target

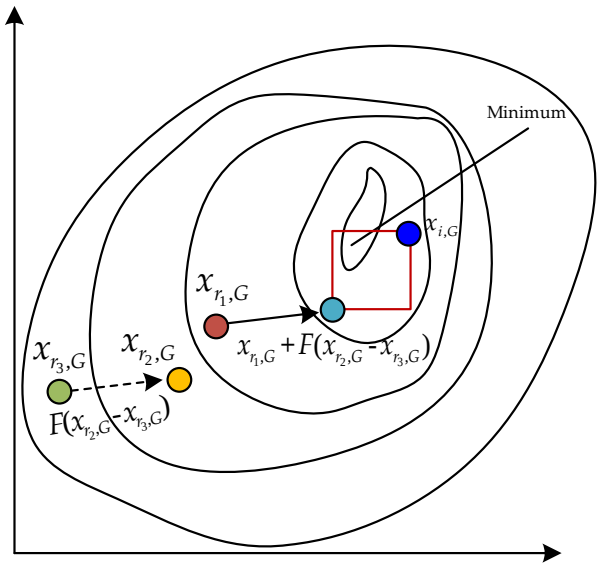


Figure 2.4: DE/rand/1/bin graphical example.

vector i at generation G. A mutant vector is created for each vector $x_{i,G}$, this is done by adding a perturbation to the existing vector using the following equation:

$$v_{i,G+1} = x_{r_1,G} + F \cdot \left(x_{r_2,G} - x_{r_3,G}\right) \tag{2.3.1}$$

Equation 2.3.1 shows $v_{i,G+1}$ as the mutant vector. Indexes r_1, r_2, r_3 are chosen randomly from the population and are different from the current vector i. F is a real and constant number $\in [0, 2]$, this parameter controls the difference amplification created by $\left(x_{r_2,G} - x_{r_3,G}\right)$.

In Figure 2.4, mutation operation in DE is graphically explained. Individuals (green, yellow and red) represents three chosen vectors. Difference between $x_{r_2,G}$ and $x_{r_3,G}$ is shown as a vector between green and yellow points. This difference is multiplied by a scalar F, adding this value to vector $x_{r_1,G}$, the mutant vector in blue is obtained.

Recombination or crossover increases diversity in perturbed vectors obtaining as a final result a trial vector:

$$u_{i,G+1} = x\left(u_{1_i,G+1}, u_{2_i,G+1}, \ldots, u_{D_i,G+1}\right) \tag{2.3.2}$$

It is calculated by Equation 2.3.3, in a way it performs recombination uniformly [12].

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1} & \text{if } (randb(j) \leq CR) \text{ or } j = rnbr(i) \\ x_{ji,G} & \text{if } (randb(j) > CR) \text{ or } j \neq rnbr(i) \end{cases} \tag{2.3.3}$$

In Equation 2.3.3 *randb* function is a random number $\in [0, 1]$. CR (crossover constant), this parameter is determined by the user, it can be seen as the crossover probability. Finally, *rnbr* chooses randomly an index $\in 1, 2, \ldots, D$, which ensures that $x_{i,G+1}$ will have at least one $v_{i,G+1}$ allele. In Figure 2.4, a red box is shown, this is where the new solutions can be placed already recombined using the mutant vector $v_{i,G}$ and the target vector $x_{i,G}$.

Finally, to decide which vector become a member of a new generation $G + 1$, trial vector $u_{i,G+1}$ is compared to the target vector $x_{i,G}$. A simple criterion can be used, if $u_{i,G+1}$ has lower fitness value that $x_{i,G}$, then target vector is set to $u_{i,G+1}$; otherwise, the old value is retained [31]. The canonical version of DE is shown in Algorithm 2.

Algorithm 2: Differential Evolution Algorithm

```
1  Create initial population x_{i,G}, i = 1, 2, ..., NP;
2  Evaluate fitness of each solution;
3  while Termination condition not satisfied do
4      for each vector x_{i,G} do
5          Select x_{r1,G}, x_{r2,G}, x_{r3,G} from population where r1, r2, r3 ≠ i ;
6          Apply x_{r1,G} + F · (x_{r2,G} − x_{r3,G}) and create mutant vector v_{i,G+1};
7          Combine target vector x_{i,G} and mutant vector v_{i,G+1} to produce trial vector
             u_{i,G+1};
8          Trial vector fitness evaluation;
9          if trial vector has higher fitness than target vector then
10             Replace target vector with trial vector;
```

2.4 Multiobjective Optimization

In previous section, main evolutionary algorithmic operations were explained considered a single objective optimization scenario. In this section, multiobjective optimization is introduced. In multiobjective problems (MOPs), solutions quality is given by the relationship between several objectives that are in conflict [12, 24]. Solving MOPs implies finding trade-offs among all the objective functions. In this kind of problems, a set of optimal

solutions is obtained instead of a single one as in the case of single-objective problems. This is because in multiobjective optimization it is not possible to find a single optimal solution which optimizes all the objective functions simultaneously. There are alternatives to avoid this problem, for example, combining fitness of each function and thus obtain a single measure, normally each function will be weighted with some fixed value, this approach is called *weight sum method* [12, 33, 24]. There are also other classic methods such as ϵ-*Constraint method* which only takes one function of a multiobjective problem and the others in conflict are taken as constraints [24, 33]. A classic method is *goal programming*, where the main idea is to find solutions close to predefined objectives for each target, if these solutions are not reached the task is to derive such objectives and attempt minimization [24]. Regarding to multiobjective evolutionary algorithms have the advantage of obtaining a set of Pareto-optimal solutions, and that no previous knowledge of the problem like weight vectors are not required. Finally, classic methods cannot find some Pareto optimal solutions when MOPs are not convex. This relates to the Pareto front's complexity and associated difficulty to find such solutions [24].

2.4.1 Basic concepts

This section explains basic concepts of multiobjective optimization.

- **Decision variables** are represented by a vector $\mathbf{x}$ with n decision variables represented by Equation 2.4.1 [33].

$$\mathbf{x} = [x_1, x_2, \ldots, \mathbf{x}_n] \tag{2.4.1}$$

- **Constraints** are imposed by environment characteristics or resources and occur in most optimization problems. They are expressed in the form of mathematical equalities or inequalities, represented in Equations 2.4.2 and 2.4.3. If the number of equality constraints is greater than the number of decision variables, the problem is over constrained so there are not enough degrees of freedom for optimization [33].

$$h_j = 0 \ , \ j = 1, 2, \ldots, p \tag{2.4.2}$$

$$g_i \leq 0 \ , \ i = 1, 2, \ldots, m \tag{2.4.3}$$

- **Objective function,** in multiobjective optimization, a set of objective functions are

used to evaluate the decision variables vector: $f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})$ where k is the number of objective functions in a multiobjective problem. The vector of objective functions $\mathbf{f}(\mathbf{x})$ is defined as [33] :

$$\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})] \tag{2.4.4}$$

- **Decision variable and objective function space** are defined by a n-dimensional space in which each coordinate axis corresponds to a component of a decision variables vector $\mathbf{x}$; and the objective function space is defined by a k-dimensional space in which each coordinate axis corresponds to a vector component $\mathbf{f}_k(\mathbf{x})$. Each point in the decision variable space represents a solution, when this vector is evaluated in the objective function, the obtained value represents a point in the objective function space which determines solution's quality. Therefore, a $F : \mathbb{R}^n \rightarrow \mathbb{R}^k$ function maps the space of decision variables to the objective function space [33]. Figure 2.5 shows this process.

2.4.2 Multiobjective optimization problem

A multiobjective optimization problem involves multiple conflicting objective functions, which must be minimized or maximized simultaneously, those functions could be subject to a number of constraints and variable bounds. Mathematically, a Multiobjective Optimization Problem (MOP) is defined as:

Definition 2.4.1. General MOP [34]:

$$\left.\begin{aligned} \text{Minimize/Maximize } \mathbf{f}_m(\mathbf{x}), m = 1, 2, \ldots, k; \\ \text{subject to } g_j(\mathbf{x}) \geq 0, j = 1, 2, \ldots, m; \\ h_k(\mathbf{x}) = 0, k = 1, 2, \ldots, p; \\ x_i^{(L)} \leq x_i \leq x_i^{(U)}, i = 1, 2, \ldots, t; \end{aligned}\right\} \tag{2.4.5}$$

with k objectives, m and p are the number of inequality and equality constraints. A solution $\mathbf{x} \in \mathbf{R}^n$ is a vector of n decision variables: $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, which satisfy all constraints and variable bounds [34, 33, 24]. The last set of constraints is called variable bounds, which restrict each upper and lower decision variable x_i. These limits are the decision variable space size. If a solution $\mathbf{x}$ satisfies all restrictions and variable bounds,

it is known as a ***feasible solution***. The set of all feasible solutions is called the ***feasible region*** [24]. It can be seen in Figure 2.5 that solutions within the blue area are feasible solutions, and their set determines a feasible area.

2.4.3 Dominance and Pareto optimality

In multiobjective optimization problems, several objectives conflict, this means that more than one optimal solution exists. These solutions are known as *Pareto-optimal* solutions. Definition of a Pareto-optimal solution is related to the domination concept as follows:

Definition 2.4.2. Pareto dominance [33]: A vector $\mathbf{u} = (u_1, u_2, \ldots, u_k)$ is said to dominate another vector $\mathbf{v} = (v_1, v_2, \ldots, v_k)$ (denoted by $\mathbf{u} \preceq \mathbf{v}$) if and only if $\mathbf{u}$ is partially less than $\mathbf{v}$, this is specified as follows: $\forall i \in \{1, \ldots k\}, u_i \leq v_i$ and $\exists i \in \{1, \ldots k\} : u_i) < v_i$.

The set of all non-dominated solutions is known as the Pareto Optimal Set (POS) and is defined as:

Definition 2.4.3. Pareto Optimal Set [33]: For a given MOP and $F(x)$, the POS $\mathcal{P}^*$ is determined by:

$$\mathcal{P}^* = \{\mathbf{x} \in \Omega \mid \neg\exists \mathbf{x}' \in \Omega \; F(\mathbf{x}') \preceq F(\mathbf{x})\} \tag{2.4.6}$$

These solutions are represented in the decision variable space. Non-dominated solutions represented in the Pareto optimal set represent the best solutions in which there is a trade-off among objectives. When mapping these solutions to the objective function space, a set called Pareto front ($\mathcal{PF}^*$) is formed, and it is defined as:

Definition 2.4.4. Pareto Front [33]:

For a given MOP, $F(X)$ and POS, $\mathcal{P}^*$, the Pareto Front $\mathcal{PF}^*$ can be expressed as:

$$\mathcal{PF}^* = \{\mathbf{u} = F(\mathbf{x}) \mid \mathbf{x} \in \mathcal{P}^*\} \tag{2.4.7}$$

In Figure 2.5, a mapping example of solutions from the Pareto optimal set to the objective function space is shown, therefore creating the Pareto front with solutions where there is a trade-off among objectives.

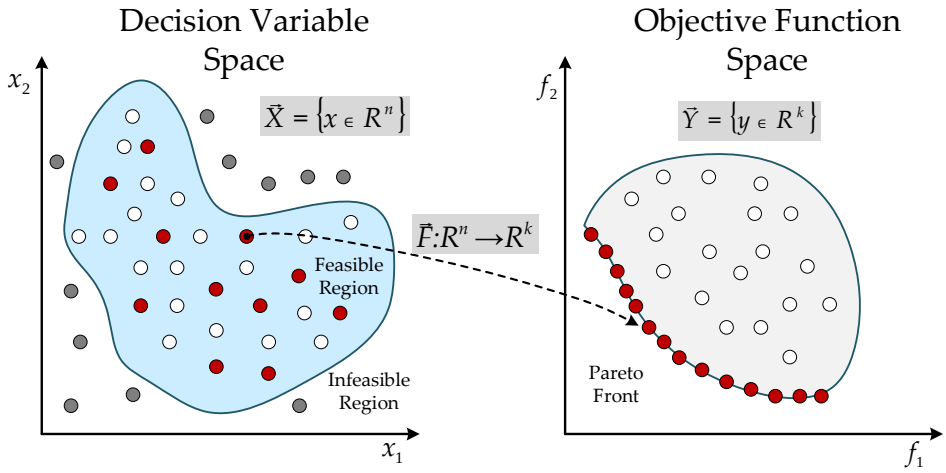


Figure 2.5: Mapping of decision variables space to the objective function space. Feasible solutions and zone are marked in blue. In the decision variable space, the Pareto optimal set is marked with red solutions and its mapping to the objective function space creates the Pareto front.

2.4.4 Special solutions

Three types of special solutions widely used in multiobjective optimization algorithms are explained. These are ideal, utopian, and nadir objective vectors, see Figure 2.6.

Ideal objective vector

For each conflicting objective exists one different optimal solution. The ideal objective vector is a vector composed by these optimal objective values [24].

Definition 2.4.5. The m-th component of the ideal objective vector Z^* is a constrained minimum solution of [24]:

$$\left.\begin{array}{l} \text{Minimeze} f_m(x) \\ \text{subject to } x \in S \end{array}\right\} \tag{2.4.8}$$

Thus, if the minimum solution for the m-th objective solution is the decision vector $x^{*(m)}$ with function value f_m^*, the ideal vector is determined as [24]:

$$Z^* = (Z_1^*, \ldots, Z_m^*) \text{ where } Z_i^* = \min f_i(\mathbf{x}) | \mathbf{x} \in Pop. \tag{2.4.9}$$

The ideal objective vector corresponds to a non-existent solution, this is because solutions of Equation 2.4.8 for each objective function need not be the same solution; More-

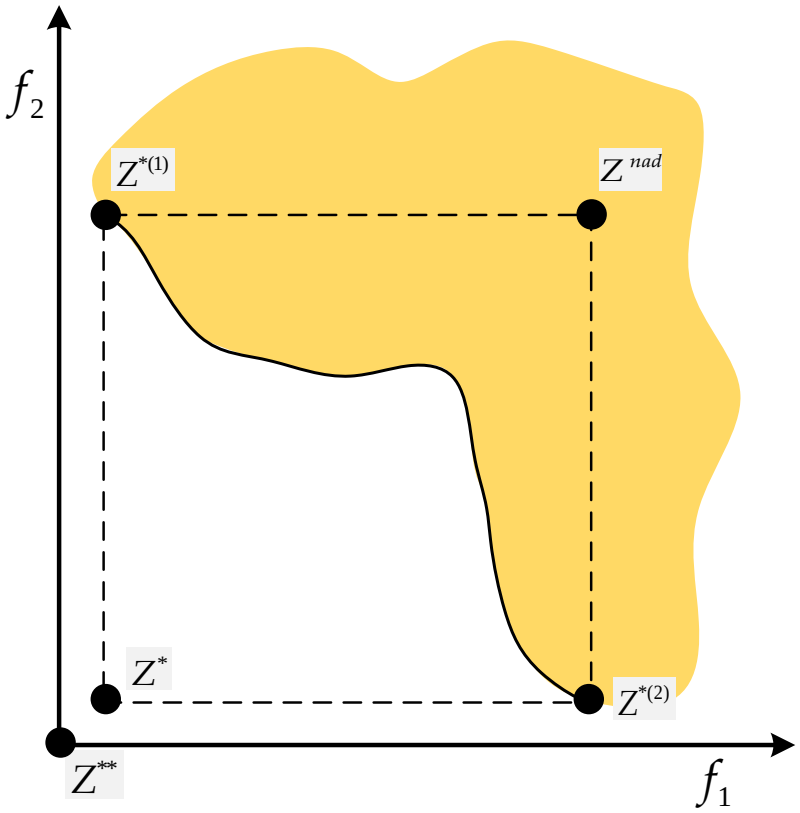


Figure 2.6: Representation of an objective function space with ideal (Z^{*}), utopian (Z^{**}), and nadir (Z^{nad}) objective vectors.

over, many algorithms require knowledge of lower bounds on each objective function to normalize objective values in a common range, in this case, the ideal vector is taken as the solution with the lowest value of solutions in the current population [24].

Utopian objective vector

The ideal objective vector is an array of all objectives functions at lower bounds. This means that for each objective function exists at least one feasible solution in the solution space, that shares an identical value with its corresponding elements in the ideal solution. Some algorithms require a solution where the objective value is strictly better not only another solution in the search space for this purpose the Utopian objective vector is defined [24]:

Definition 2.4.6. In a Utopian objective vector Z^{**} each component is sightly smaller than the ideal objective vector, or $Z_i^{**} = Z_i^* - \epsilon_i$ with $\epsilon > 0$ for all $i = 1, 2, \ldots, M$

Similar to the ideal objective vector, the Utopian objective vector also represents a non-existent solution [24].

Nadir objective vector

Refers to the vector opposite the ideal objective vector which represents the lower limit of each objective in the entire search space. The Nadir objective vector represents an upper limit of each objective vector in the Pareto optimal set, but not throughout the search space [24]. In some algorithms, such as MOEA/D, the nadir point is represented as the solution which is the upper limit [28], the maximum solution in a population and is defined as:

$$Z^{nad} = (z_1^*, \ldots, z_m^*) \text{ where } n_i^* = \max f_i(\mathbf{x}) | \mathbf{x} \in P. \tag{2.4.10}$$

2.5 Multiobjective evolutionary algorithms

The multiobjective evolutionary algorithms in comparison with the classic mathematical programming methods mentioned briefly in Section 2.4, have certain characteristics and advantages that make them applicable to solve MOPs [24], therefore, in this section different evolutionary algorithms approaches to target multiobjective problems are reviewed. These algorithms can be divided into three main paradigms [35], Pareto based MOEAs, decomposition based MOEAs and indicator based MOEAs. In this thesis, Pareto based MOEAs and decomposition based MOEAs are explained in detail.

2.5.1 Pareto-based MOEAs

Pareto-based MOEAs use a dominance based ranking scheme and combine elitist strategies such those that converge to a global optimal in some problems [12]. Pareto and elitist strategies lead the way or set the basis for one of the most important algorithmic approaches in the area: NSGA-II algorithm proposed by Deb et. al. [21]. Pareto based MOEAs have in common the use of Pareto dominance with some diversity criteria based on secondary ranking, some algorithms of this class are Multiobjective Genetic Algorithm (MOGA) [36], which was the first MOEA, Pareto Archived Evolutionary Strategy (PAES) [37] uses a mesh in the objective function space to ensure that all regions are visited,

Strength Pareto Evolutionary Algorithm (SPEA) [38] which uses a different criteria based on dominance, it also ranks individuals depending on how many individuals dominate and by how many other individuals dominate it, as well as it makes use of clustering, in its second version (SPEA-2) [39] this algorithm improves both criteria already described.

NSGA-II

Non-dominated sorting Genetic Algorithm (NSGA-II) was proposed by Deb et. al. in 2002 [21], among its main characteristics is the use of elitism, as well as the use of a mechanism of diversity and focus on non-dominated solutions. This is the improved NSGA version [40] that focuses on reduction of complexity of non-dominated sorting, more efficient use of elitism, and parameters reduction. NSGA-II starts with a random population, each offspring is created using two parents selected through binary tournament method, parents are recombined using SBX operator and mutated using polynomial mutation [21]. At this point ofspring set Q is combined with current population $P \cup Q$. The best μ individuals are selected from $P \cup Q$ concerning the front to which it belongs, this ranking is obtained with non-dominated sorting (See Algorithm 4) which ranks solutions by corresponding fronts, thus obtaining a measure of quality to evaluate different solutions. If the set of μ solutions is greater than the population size, then a ranking process called *crowding distance* (see Algorithm 3) is applied. The larger distance this metric is better ranked the solution, because it maintains the diversity in the population. Crowding distance is calculated as the distances average of neighboring solutions, thus generating a cuboid in the objective function space. Finally, μ solutions are passed on to the next generation, repeating the process mentioned above, NSGA-II is described in Algorithm 5. Recently NSGA-II has been modified to address many objectives in its NSGA-III [4] version, which combines NSGA-II ideas with decomposition approaches.

Algorithm 3: Crowding Distance [21]

```
1 l = |I|;
2 I[i]_distance = 0;
3 for each objective m do
4 |   I = sort(I, m);
5 |   I[1]_distance = I[l]_distance = ∞;
6 |   i = 2 to (l − 1) I[i]_distance = I[i]_distance + (I[i + 1].m − I[i − 1].m)/(f_m^max − f_m^min);
```

Algorithm 4: Non-dominated sorting [21]

```
for each p ∈ Population P do
    S_p = ∅, n_p = 0;
    for each q ∈ P do
        if p ≺ q then
            S_p = S_p ∪ {q};
        else
            if q ≺ p then
                n_p = n_p + 1;
        if n_p = 0 then
            p_rank = 1;
            F_1 = F_1 ∪ {p};
while F_i ≠ ∅ do
    Q = ∅;
    for each p ∈ F_i do
        for each q ∈ S_p do
            n_q = n_q − 1;
            if n_q = 0 then
                q_rank = i + 1;
                Q = Q ∪ {q};
    i = i + 1;
    F_i = Q;
```

Algorithm 5: NSGA-II [21]

```
Create initial population P_t;
Evaluate fitness of each solution;
Apply non-dominated sorting to rank the solutions;
while Termination condition not satisfied do
    Offspring population Q_t = ∅;
    for each solution in P_t do
        Select two parents using binary tournament;
        Recombine the parents using SBX and generate a child r;
        Apply mutation on r generating q;
        Q_t = Q_t ∪ q;
    Apply Algorithm 4 to rank P_t ∪ Q_t population obtaining F fronts;
    if |F_1| + |F_2|, ..., |F_i| = |P_t| then
        Copy the solutions of these fronts to the new population P_{t+1};
        P_{t+1} = ⋃_i F_i;
    else
        Determine the front H of F_i that is greater than |P_t|, to this last front apply Crowding
          Distance (Algorithm 3) and add to P_t the solutions with the higher Distance;
        P_{t+1} = (⋃_i F_i) ∪ H;
```

MOCell

Cellular genetic algorithm for multiobjective optimization MOCell [9] was proposed by Nebro et al. in 2009, this approach uses as base the cellular genetic algorithm and adding different improvements to work in the multiobjective field, this begins generating a toroidal mesh in which the solutions are put, later, it follows the normal steps of a genetic algorithm (Selection, crossing, mutation and evaluation), when replacing the worst solutions, it is based on the NSGA-II algorithm. If the solutions are non-dominated, then the Crowding Distance is used. as final steps it takes an external file which saves the best solutions these are the non-dominated solutions, as final step MOCell makes use of a feedback mechanism which is to take from the external file and re-insert again the population, this feedback mechanism is one of the most important of the algorithm as it promotes diversity over generations [2]. The pseudocode of MOCell algorithm is shown in Algorithm 6, and a general diagram of the evolutionary process in Figure 2.7.

In Section 2.7 it is extended the background about the parallel evolutionary algorithms, their different structures, properties, and update mechanisms.

Algorithm 6: MOCell

```
1  Create initial population P;
2  while Termination condition not satisfied do
3      for each individual i in P do
4          N is the neighborhood of individual i;
5          Select two parents from N using binary tournament;
6          Recombine the two parents using SBX and create an offspring y;
7          mutate the offspring y using polynomial mutation to obtain y';
8          Insert solution y' in an auxiliary population only if it is non dominated;
9          Insert solution y' to an external archive only if is non dominated.
10     updates the population using the auxiliary population;
11     Applies feedback mechanism using the external archive;
12     if External archive size > population size then
13         Remove the worst solutions based on crowding distance criteria;
```

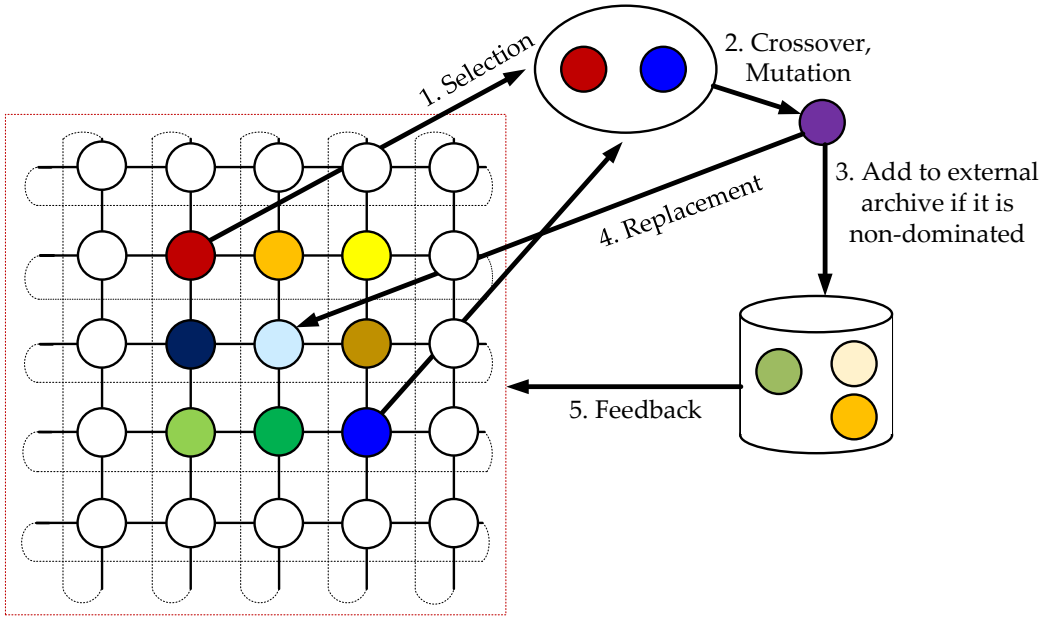


Figure 2.7: Scheme of evolutionary process of MOCell algorithm.

Generalized Differential Evolution 3

Generalized Differential Evolution (GDE3) is the third version of the algorithm proposed by Kukkonen and Lampinen [41] in 2005, as a version of the differential evolution algorithm to solve multiobjective problems with constraints, GDE3 makes use of the DE/rand/1/bin scheme, to be used with problems with M objectives and K constraints, in this version as well as in the second one it makes use of the crowding distance, GDE3 is ahead of its two previous versions which were very sensitive to the chosen parameters in addition to the use of the non-dominated sorting. GDE3 modifies the basic selection rules of differential evolution which have their base in Deb's rules or Constraint dominance principle (see Section 2.6) that are used to define the comparison between trial and target vectors (see line 9 of Algorithm 7) , in the same way, that NSGA-II when the population grows, is necessary to truncate, GDE3 makes use of the Crowding distance and non-dominance (see line 13 of Algorithm 7, *CD* means Crowding Distance), which allows an elitist effect by selecting the best solutions in the population and discarding the worst ones [41, 33], the pseudocode of GDE3 is shown in Algorithm 7.

Algorithm 7: GDE3

Create initial population of size NP;
while *Termination condition not satisfied* **do**
  for *each target vector* x_i *in population* **do**
    Mutate and recombine using Differential Evolution operator obtaining the trial vector u_i;
    if *trial vector* $u_i \preceq$ *target vector* x_i **then**
      $x_{i,G+1} = u_i$;
    else
      $x_{i,G+1} = x_i$;
    if $\forall_j : g_j(u_i) \leq 0 \wedge (x_{i,G+1} == x_{i,G}) \wedge x_{i,G} \npreceq u_i$ **then**
      $m = m + 1$;
      $x_{NP+m,G+1} = u_i$
  while $m > 0$ **do**
    select $x \in \{x_{1,G+1}, x_{2,G+1}, \ldots, x_{NP+m,G+1}\} : \begin{cases} \forall_i x \npreceq x_{i,G+1} \\ \wedge \\ \forall (x_{i,G+1} : x_{i,G+1:} \npreceq x)\ CD(x) \leq CD(x_{i,G+1}) \end{cases}$
    remove x from population;
    $m = m - 1$;

2.5.2 MOEAs based on Decomposition

An inevitable problem when dealing with MOPs of more than 3 objectives, is that dominance begins to be ineffective, therefore, the idea of rank Pareto's front is not useful. Therefore, decomposition-based methods have been adapted to MOEAs, these methods can deal with problems of three or many objectives, but also work with MOPs, giving a reliable and powerful alternative for this kind of problems, one of the most important algorithms of this approach is MOEA/D (Multiobjective Evolutionary Algorithm based on Decomposition) developed by Zhang and Li [18, 12, 19].

MOEA/D

Multiobjective Evolutionary Algorithm Based on Decomposition (MOEA/D) is an algorithm which shares some characteristics of the weighted-sum approach and population-based algorithms. MOEA/D starts by distributing a set of λ weight vectors in the objective function space, then creates an array of the T closest vectors using the Euclidean distance between vectors, thus creating neighborhoods. MOEA/D and its variants demonstrate a performance equivalent to the Pareto based algorithms in problems with few objectives and better performance in problems with 5 or more objectives [12, 18]. The Canonical

MOEA/D algorithm use Tchebycheff decomposition defined as follows:

$$\min g^{te}(\mathbf{x}|\lambda^j, z^*) = \max_{1\leq i\leq m} \frac{1}{\lambda_i^j}|f_i(\mathbf{x}) - z_i^*| \tag{2.5.1}$$

MOEA/D algorithm is shown in Algorithm 8, in the first lines (1-4) λ reference vectors are established and neighborhoods are created using as a criteria nearest T vectors, and ideal point Z is calculated. The main cycle iterates over all individuals within the population. In line 7 two parents are chosen, these are taken from the neighborhoods created and stored in $B(i)$. $B(i)$ is iterated and two random parents are chosen from this structure, then SBX operator and polynomial mutation are used to generate the offspring, it is worth mentioning that here only one child is generated not two, in the final parts of the algorithm, the value of Z is updated again and the aggregation values of the two parents are calculated using the λ reference vectors, likewise aggregation value of child y^i is calculated, finally, it is replaced with a simple criterion, if the offspring y^i has a smaller aggregation value of one of the parents this is replaced otherwise the parent remains and the population is not modified.

Algorithm 8: MOEA/D

1 $\lambda^i = (\lambda_1^i, \ldots, \lambda_m^i)^T, i = 1, \ldots, N_p$;
2 $B(i) = \{i_1, \ldots, i_t\}$, where $\lambda^{i_1}, \ldots, \lambda^{i_T}$ are the T closest weight vectors to λ^i.;
3 $P = \{\mathbf{x}^1, \ldots, \mathbf{x}^{N_p}\}$;
4 Set Z using equation 2.4.9;
5 **while** $k \leq T_{max}$ **do**
6 **for** $i = 1$ **to** $size(P)$ **do**
7 $P = B(i, randperm(B))$;
8 Generate y from $\mathbf{P(1)}$ and $\mathbf{P(2)}$ by GA operator;
9 Polynomial mutation on y to new solution y^i;
10 Update Z using equation 2.4.9;
11 $g_{pop} = g^{te}(P|\lambda^P, z^*)$;
12 $g_y = g^{te}(y^i|\lambda^P, z^*)$;
13 $Population(P(g_{pop} \geq g_y)) = y^i$;

There are many variants of MOEA/D, one of the most popular and widely used as a basis for new algorithms based on decomposition is MOEA/D-DE [19], this is a modified version that uses differential evolution operations. MOEA/D-DE uses Differential Evolution such as crossover operator. Also, they add other parameters to the original MOEA/D algorithm, these parameters are κ and nr. The first parameter κ is the probability of selecting from two sets, from neighborhood $B(i)$ or from the whole population, the selection of

parents is done randomly in either set, this parameter is commonly 0.9, this means that the probability of choosing parents from neighborhood is much larger than the entire population, this to increase the exploration probability of the algorithm. The *nr* parameter is fixed as the number of replacements in the updating solution step, in the original algorithm, the number of replacements is commonly set to 2. MOEA/D-DE uses Equation 2.5.2 as the crossover operator.

$$\overline{y}_k = \begin{cases} x_k^{r_1} + F \times (x_k^{r_2} - x_k^{r_3}) & \text{with probability } CR \\ x_k^{r_1} & \text{with probability } 1 - CR \end{cases} \tag{2.5.2}$$

Equation 2.5.2 is a variation of the original equation in Differential Evolution Algorithm. As a mutation operator, MOEA/D-DE algorithm uses polynomial mutation 2.1.9. in Algorithm 9 MOEA/D-DE is shown.

Algorithm 9: *MOEA/D − DE*

$\lambda^i = (\lambda_1^i, \ldots, \lambda_m^i)^T, i = 1, \ldots, N_p$;
$B(i) = \{i_1, \ldots, i_t\}$, where $\lambda^{i_1}, \ldots, \lambda^{i_T}$ are the T closest weight vectors to λ^i;
$P = \{\mathbf{x}^1, \ldots, \mathbf{x}^{N_p}\}$;
Set Z using equation 2.4.9;
while $k \leq T_{max}$ **do**
  for $i = 1$ **to** $size(P)$ **do**
    if $rand < \kappa$ **then**
      $P_s = B(i, randperm(B))$;
    else
      $P_s = P(randperm(P))$;
    Generate y from $\mathbf{P_s(1)}, \mathbf{P_s(2)}t$ and $\mathbf{P_s(3)}$ by DE operator using equation 2.5.2;
    Polynomial mutation on y to new solution y^i;
    Update Z using equation 2.4.9;
    $g_{pop} = g^{te}(P|\lambda^P, z^*)$;
    $g_y = g^{te}(y^i|\lambda^P, z^*)$;
    while $P_s \neq \emptyset$ *or* $c \neq nr$ **do**
      j = random number in P_s;
      if $g_Y \leq g_{pop}$ **then**
        $P(P_s(j)) = y_i$;
        $c = c + 1$;
      $P_s(j), g_y(j), g_{pop}(j) = \emptyset$;

2.6 Constraint Handling techniques

Handling constraints in MOEAs is an important issue that requires special attention, mostly when dealing with real world problems, constraint Handling techniques (CHTs) must be incorporated into the search process in evolutionary algorithms. [33]. A common way to deal with constraints is converting inequality constraints to equality constraints, as follows:

$$h_j(\mathbf{x})' \equiv \delta - |h_j(\mathbf{x})| \geq 0 \tag{2.6.1}$$

Moreover, a number of CHTs use constraint violation sum criterion, defined next [28]:

$$\phi(\mathbf{x}) = \sum_{i=1}^{q} |min(g_i((\mathbf{x}), 0)| + \sum_{j=1}^{p} |min(h_i((\mathbf{x})', 0)| \tag{2.6.2}$$

Deb proposed a CHT based only on the sum of constraint violations. In NSGA-II [21] this is extended to multiobjective problems, this CHT is called Constraint Dominance Principle (CDP) this compares two solutions (x and y) based on the follow rules [42].

1.- x is feasible and y is infeasible

2.- both x and y are infeasible and x has a less sum of constraint violation than y.

3.- both x and y are feasible and x dominates y

Another method commonly used is Stochastic Ranking (SR), in this CHT, a predefined probability is used to balance dominance comparison of two solutions according to its objective function or sum of constraint violation value [43]. Other method is known as ϵ-constrained which is similar to CDP, the only difference is that a solution is treated as a feasible solution if its constraint violation sum is less than a given threshold ϵ; in other words, this CHT transforms the constrained problem into an unconstrained one. When ϵ is equals to zero, epsilon CHT is equivalent to CDP. Finally, another widely used method to penalize the objective function which involves weights the sum of constraints violations [34]. There are several different types of penalty functions, which includes death penalty, static penalty, dynamic penalty, and adaptive penalty.

The main issue with penalty functions for constraint handling is that ideal penalty factors can not be known in advance for an arbitrary Constraint MOP, therefore, tuning these penalty factors requires prior knowledge of the problem [43]. Constraint handling

methods described here can be adapted to any type of MOEAs as they only require small modifications within the algorithm.

2.6.1 Push and Pull Search

The Pull and Push Search (PPS) was developed to target CMOPs [29]. PPS follows two stages: first, a CMOP is approached without constraints trying to reach the unconstrained Pareto front (See Figure 2.8a, 2.8b, and 2.8c where the problem is taken without constraints, the solutions approach the unconstrained Pareto front), and secondly stage, it implements a CHT which assures a better close approach to the constrained Pareto front because solutions are already close, this is seen in Figure 2.8d, where the problem becomes constrained and the solutions are already closer to the constrained Pareto front. A strategy for switching between both stages follows the next condition:

$$rk \equiv \max\{rz_k, rn_k\} \leq \rho \tag{2.6.3}$$

where rk represents the maximum change rate between ideal and nadir points, see Equations 2.4.9 and 2.4.10, in l generations, rz_k and rn_k are defined next:

$$rz_k = \max_{i=1,\ldots,m} \{\frac{|z_i^k - z_i^{k-l}|}{\max\{|z_i^{k-l}|, \Delta\}}\} \tag{2.6.4}$$

$$rn_k = \max_{i=1,\ldots,m} \{\frac{|n_i^k - n_i^{k-l}|}{\max\{|n_i^{k-l}|, \Delta\}}\} \tag{2.6.5}$$

where z^k and n^k are ideal and nadir points in $k-$th generation, rz_k and rn_k are two points within $[0, 1]$, Δ is a very small positive number to avoid denominators equal zero, thus Δ is set to $1e-6$.

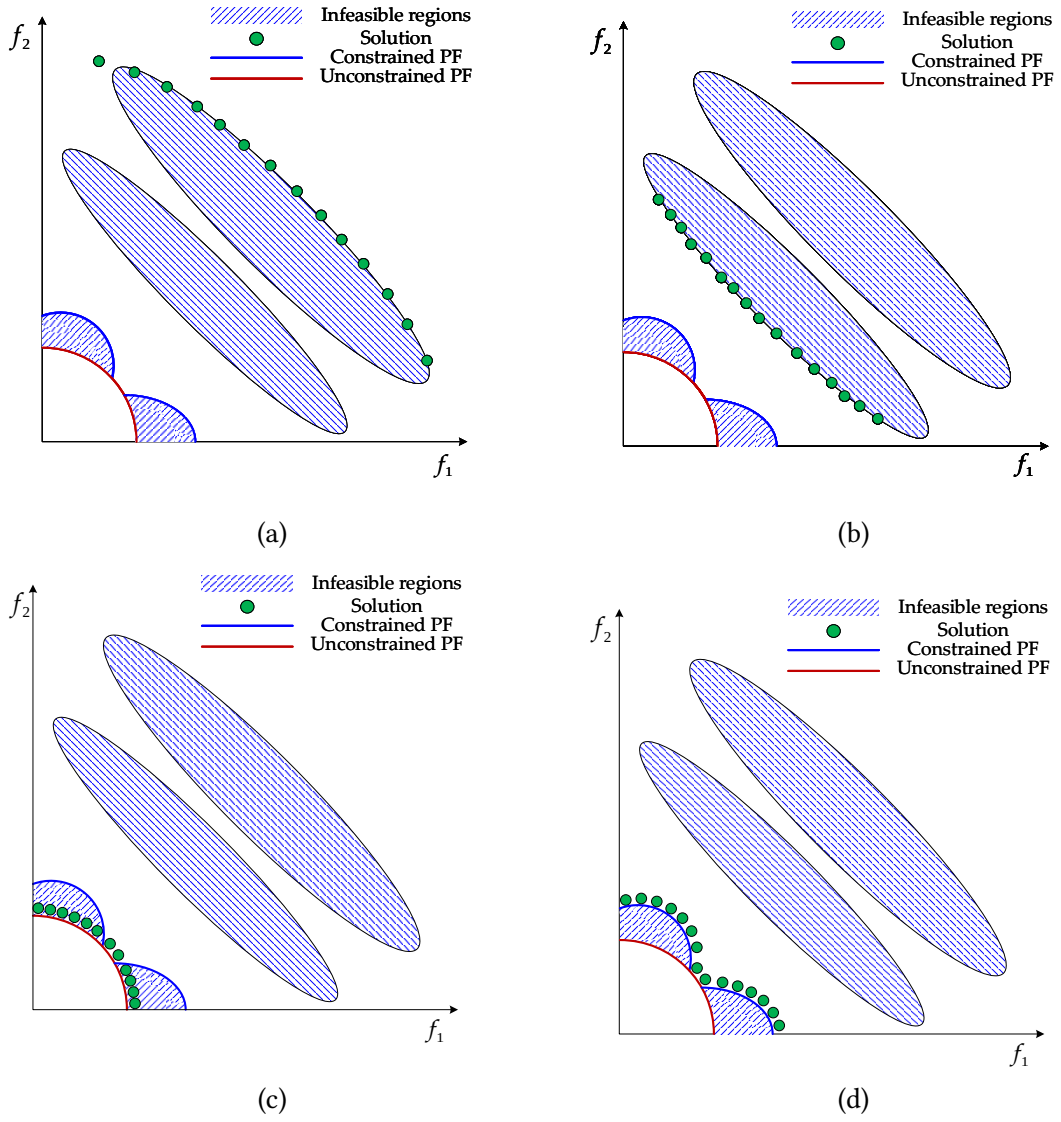


Figure 2.8: In the first three subfigures, the process of pushing towards the unconstrained Pareto front is observed, in subfigure (d) the process of pulling towards the constrained Pareto front observed.

2.6.2 Improved Epsilon

Improved Epsilon [27] defines a flexible ϵ parameter which is dynamically reduced as generations elapse until reaching a zero value. It allows a looseness value of the total sum of constraints violation for considering offspring as feasible solutions or not. This dynamic ϵ is updated every generation according to Equation 2.6.6 and its value decreases to zero as it is considered that, in final generations, solutions had reached feasible regions within the landscape.

$$\epsilon = \begin{cases} (1-\tau)\epsilon(k-1) & \text{if } rf_k < \alpha \\ (1+\tau)\phi_{max} & \text{if } rf_k \geq \alpha \end{cases} \tag{2.6.6}$$

where rfk is the population's feasibility ratio in generation k, τ is within $[0, 1]$ range and controls constraints speed relaxation by multiplying the maximum sum of constraints violations in a population. An α parameter controls the searching preference between feasible and infeasible zones, ϕ_{max} is a solution with the maximum overall constraints violations sum in all elapsed generations and is updated every generation by Equation 2.6.7.

$$\phi_{max} = \max(\phi(P)) \tag{2.6.7}$$

2.7 Parallel evolutionary algorithms

The term parallel and distributed EAs has been used indistinctly in Evolutionary Computation. A parallel EA tries to find better solutions while reducing the number of evaluations and therefore processing time, different to traditional EAs. An EA is parallelized either by distributing the objective function calculation or by dividing the entire population in sub-populations among processing units while defining migration policies for their interaction [44, 45, 46].

Parallel EAs can be assessed at an algorithmic level without necessarily deploying a number of processing units. In this scenario, parallel EAs performance is different to a standard panmictic EAs. At an implementation level there are different architectures that have been used for assessment [47, 48] such as Single-Instruction Multiple-Data (SIMD) or Multiple Instruction, Multiple Data (MIMD). Moreover, interconnected computers has taken relevance in addition to powerful tools such as Message Passing Interface (MPI), Java-Remote Method Invocation (RMI), Common Object Request Broker Architecture

(CORBA), as well as the use of the internet for communication among dedicated machines. Currently the use of large clusters of computers and technologies such as Hadoop with programming languages such as MapReduce make the task easier at an implementation level [49].

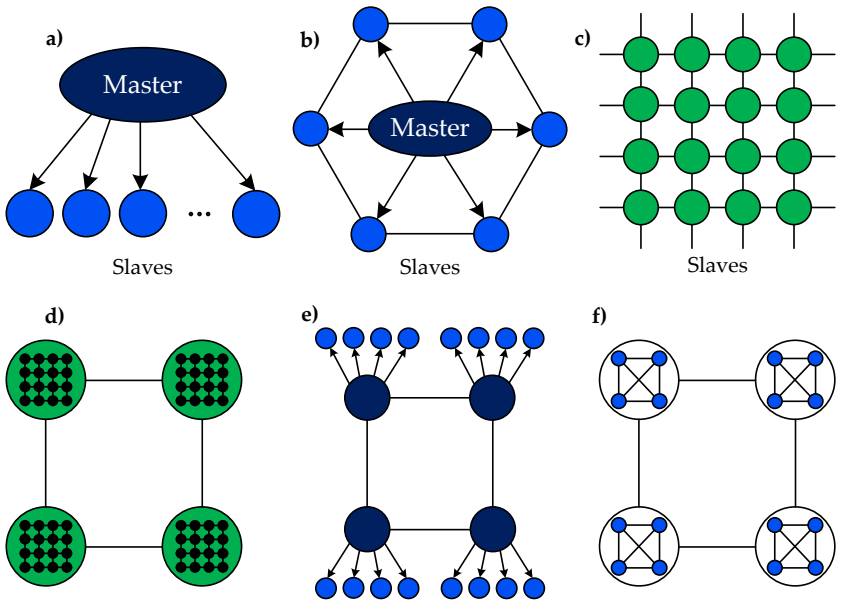


Figure 2.9: a) Master-slave model , b) Distributed or island model, c) Diffusion or cellular, and d), e), f) Multiple type of hybridization models.

The most common models for parallel EAs are master-slave, islands, diffusion or cellular, and the hybrid model which can be seen as a combination of the previous three, see Fig. 2.9. The master-slave model or global parallelization is a scheme with a central processing unit which executes selection among solutions and associates those to evolve within a slave processing unit. Thus, evolutionary operations such as recombination, mutation and objective function evaluation are executed in parallel. This model is fast, especially for the time consumed in calculating the objective function [50]. However, other EAs models use structured populations, an example of those are distributed and cellular models.

Distributed scheme divides the population into islands and these are executed in parallel. These islands exchange information during the search in order to promote population diversity among sub-populations. In cellular schemes, sub-populations are typically composed by a single individual which interacts with the rest of the population through a grid topology and predefined neighbourhoods. Neighborhoods are overlapped which implicitly defines a migration policy and allows a smoother diffusion of solutions throughout

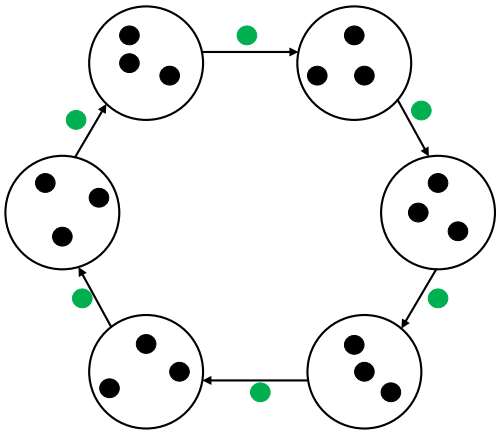


Figure 2.10: Distributed model also called island model.

the entire population which is decentralized on a toroidal grid. Cellular schemes are compatible in an ideal scenario with massively parallel machines, but sequential execution is possible on CPUs or in parallel processors such as graphics units (GPUs) [8, 50, 44, 46]. All these ideas can be used in single and multi objectives approaches, because structures can be adapted to a different type of algorithms commonly EAs and MOEAs own the same structure. An example can be cMOGA algorithm [51], this algorithm is a canonical version of a cellular GA in the multiobjective Domain.

2.8 Distributed evolutionary algorithms

In distributed EAs, the population is structured in subpopulations, relatively isolated from each other, this type of paradigm is known in different ways: island model, coarse-grain model or multi-deme model. Those subpopulations or islands evolve independently, and occasionally certain solutions migrate from one island to another [48, 2, 45], this is exemplified in Figure 2.10. Different criteria can modify a distributed model, they are focused on migration policies [52, 45]:

- Migration gap decides how often a change is made among islands. It can be activated in each subpopulation periodically or using some migration probability to decide when migration take or does not take place. Migration frequency implies slower or faster information spreading. Slower migration increases exploitation of specific regions in the landscape while faster migration promotes exploration throughout the search space.

- Emigration policy decides what actions are taken when migrants are sent to another island. Multiple actions can be taken, such as removal from the sending island, to make a copy of an immigrant. Also, there are several ways of selecting migrants, such as selecting the best, the worst, or at random.

- Immigration policy determines immigrants actions. They can replace the worst individuals in the receiving population. They can also be replaced at random or according to some condition such as replacing the most similar one.

- Migration rate specifies the number of migrants that will be sent to a different island.

- Migration topology defines the type of topology used. It can be a directed or undirected graph. The most common topologies are one-way rings or two-way rings, toroids or hypercubes.

2.9 Cellular evolutionary algorithms

Cellular evolutionary algorithms (cEAs) are a special case of the islands model with a more fine grained form of parallelization, This model is also called the diffusion model. The main feature in cEAs is that each island holds a single individual [46, 45]. An individual position is called a *cell* and it is allowed to recombine with its neighbors. Neighborhoods are defined by a mesh structure which is commonly squared, see Subfigure 2.11b, but other structures can also be used. Neighborhoods are overlapped which makes solutions to spread throughout grid [52, 46]. To replace individuals two general strategies are used. Cells can be updated synchronously or asynchronously. Special cases of asynchronous update are defined as follows [2, 52, 45]:

- Uniform choice: the cell to be updated is chosen at random.

- Fixed-line sweep or Line sweep: the cells are updated sequentially line by line.

- Fixed random sweep: the cells are updated sequentially according to a fixed order, this order is established by an exchange of the cells.

- New random sweep: this strategy is such as a fixed random sweep only after each update a new random swap is generated.

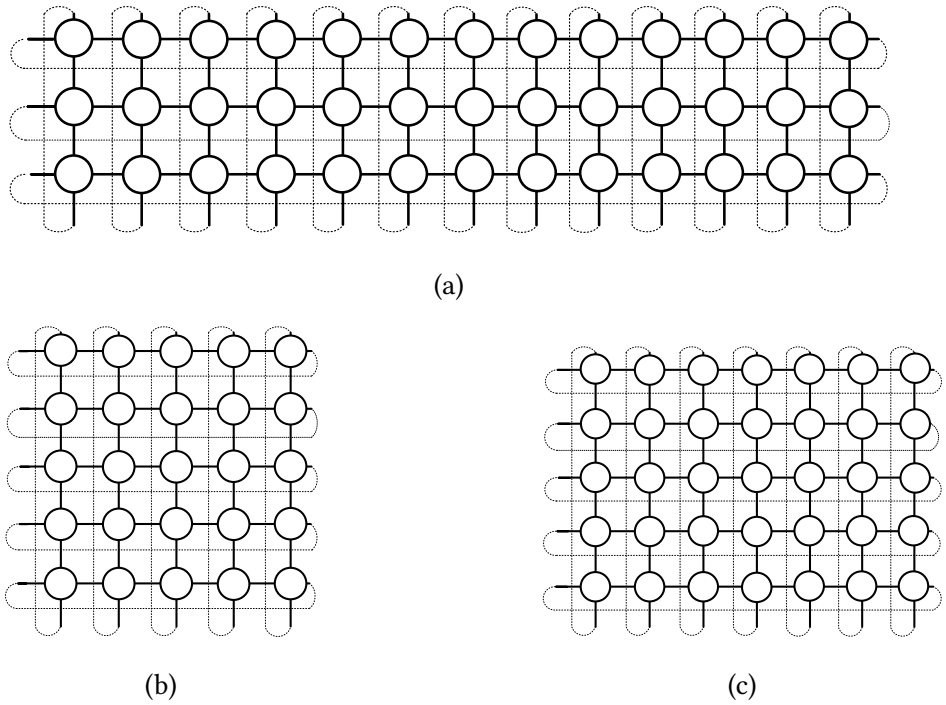
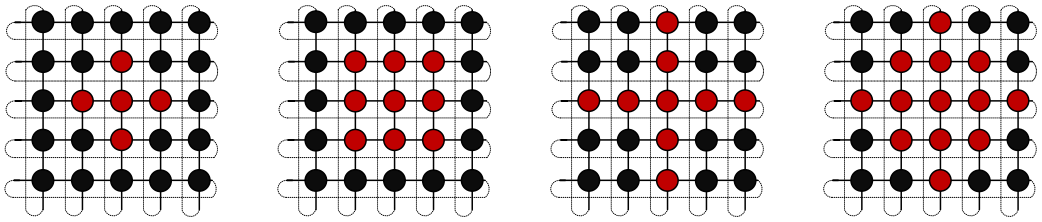


Figure 2.11: Three different types of mesh topologies: (a) Narrow, (b) Square, and (c) Rectangular. The individuals in the figure are represented by circles.

In this thesis, three updating types are used: synchronous, line sweep and asynchronous. Last type updates cells immediately after being evaluated. It could be said that it is the highest level of asynchronism, see Section 2.9.1 for more details.

Rectangular and narrower topologies have been implemented with a different impact on solutions spreading throughout the mesh and therefore in the searching process. [52, 45, 2]. Figure 2.11 shows different mesh topologies. Each point in the mesh has a neighborhood that is overlapped by nearer neighborhoods. All neighborhoods have the same size and the same form, in Figure 2.12, four types of neighborhoods commonly used in cEAs are shown. These neighborhoods are divided in two types Ln (linear) that is formed by n nearer neighborhoods in an axial direction (North,East,West,South); whereas Cn (compact) is used to design neighborhoods that contain $n-1$ nearest individuals considering diagonal directions; two of the most used are L5 also called *Von Neumann* or NEWS neighborhood and C9 also known as *Moore* neighborhood [2, 48].

Concerning selection methods designed specifically for cellular schemes, two variants applying tournament selection were proposed, the first called Anisotropic selection [53], which was developed in conjunction with the Von Neumann Fuzzy neighborhood. It de-



(a) (b) (c) (d)

Figure 2.12: Subfigure (a) shows the neighborhood L5 or Von Neumann, (b) shows the neighborhood C9 or Moore, (c) and (d) show two variants, the neighborhood L9 and the neighborhood C13.

fines a set of probabilities for k cells, then applies a k tournament and the winner replaces the individual with a probability of 0.5 if it is equal. Besides, Centric selection [54] was developed, it introduces a probability that selects the center of a neighborhood. This type of selection can be used in self-adaptive algorithms, with the advantage of modifying the exploration and exploitation trade-off by changing the probability of selecting only the center of each neighborhood [54]. Focusing on cellular schemes, solutions replacement can vary, in multiobjective scenarios, since the 'best solution' concept changes according to how a solution is measured i.e Pareto or decomposition. For example, MOCell [9] occupies a replacement scheme for the non-dominated solutions (this can be seen as a modification of the replacement worst scheme) and an elitist approach, which is used most of the time in this type of algorithms, these are carried out at the neighborhood level.

2.9.1 Updating criteria Cellular EAs

In cellular EAs in specific cellular genetic algorithms, there are several ways for population updating. In this research, three of those approaches are explored: synchronous (SY), line sweep (LS) and asynchronous (AS) updating [2]. Each updating mechanism has an effect on how new individuals interact with the rest of the population throughout the toroidal grid,

SY: this criterion expects all solutions are evolved before updating is carried out. Thus, solutions are temporarily stored and will be replaced throughout the grid at the end of every generation. In this case, ω variable contains all the indexes and individuals that will be updated and P is the Population which will be updated.

Algorithm 10: $UpdateSynchronous(P, \omega)$

```
for i=1 to size(ω) do
    {y, index} = ω(i);
    P(index) = y;
    Result: P
```

LS and AS versions are in the main cycle because solutions are updated in a constant way in contrast with SY criterion that waits until all solutions are evolved and updates the entire grid at the end of each generation. More details on these criteria are provided next.

LS: having an $N \times N$ population's grid, after first N individuals in a row are evolved, updating is carried out. This criterion stores temporarily new solutions for every row. Different from synchronous updating, LS introduces new solutions after evolving every grid's row which necessarily impacts the whole searching process, in Algorithm 11, the variable i represents the current individual in the mesh.

Algorithm 11: $UpdateLineSweep(P, \omega, i, N)$

```
if mod(i, size(N)) == 0 then
    for i = 1 to size(ω) do
        {y, index} = ω(i);
        P(index) = y;
    ω = ∅;
Result: P
```

AS: after every individual in the grid is evolved, updating takes place. Thus, new solutions are introduced immediately after replacement decision is made, therefore, it is not necessary to temporarily store information. AS updating strongly impacts the evolutionary process because it constantly introduces new solutions.

Algorithm 12: $UpdateAsynchronous(P, \omega)$

```
if ω ≠ ∅ then
    {y, index} = ω;
    P(index) = y;
    ω = ∅;
Result: P
```

2.10 Performance assessment of MOEAs

Different from single-objective optimization, where the quality of a solution can be defined using the objective function values: the smaller (to minimize) or the larger (to maximize) value corresponds to a better solution, in multiobjective optimization, other aspects should be considered to evaluate the performance of MOEAs

To evaluate the performance of different MOEAs on a given problem, the algorithm is executed a number of times, and resulting solutions known as Pareto front approximations ($\mathcal{PF}_{known}$), are compared in two aspects: (i) Solution accuracy determines how similar an evolved solution is to the true Pareto front ($\mathcal{PF}_{true}$) and (ii) Solution diversity, e.g. to evaluate how well the solution is distributed in the solution space [6, 9]. Therefore, to assess the performance of the MOEAs, several performance measures have been proposed which considered the two above issues.

- **Generational Distance**. This metric (GD) reports how far, on average the $\mathcal{PF}_{known}$ is from $\mathcal{PF}_{true}$. It is defined as

$$GD = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n} \tag{2.10.1}$$

  where n is the number of elements in the $\mathcal{PF}_{known}$ and d is the Euclidean phenotypic distance between each member [33, 6].

- **Inverted Generational Distance**. This metric (IGD) evaluates the performance related to convergence and diversity simultaneously, this metric represents the average distance from $\mathcal{PF}_{true}$ to $\mathcal{PF}_{known}$ [10]. It is defined as:

$$\begin{aligned} IGD &= \frac{\sum_{y^* \in \mathcal{PF}_{true}} d(y^*, \mathcal{PF}_{true})}{n} \\ d(y^*, \mathcal{PF}_{true}) &= min_{y \in \mathcal{PF}_{true}} \sqrt{\sum_{i=1}^{m} (y^* - y_i)^2} \end{aligned} \tag{2.10.2}$$

- **Hypervolume**. This metric (HV) reflects the closeness between $\mathcal{PF}_{known}$ and $\mathcal{PF}_{true}$. A large HV means that the $\mathcal{PF}_{known}$ set is closer to the $\mathcal{PF}_{true}$. HV corresponds to the non-overlapping volume of all hypercubes formed by reference point z and every vector in the $\mathcal{PF}_{known}$. HV with a larger value represents better per-

formance with respect to both diversity and convergence. It is defined as follows:

$$HV = \bigcup_{i=1}^{Q} \{vol_i | vec_i \in \mathcal{PF}_{true}\} \tag{2.10.3}$$

where vec_i is a non-dominated vector from $\mathcal{PF}_{known}$, Q is a set of $\mathcal{PF}_{known}$ solutions and vol_i is the hypercube's volume formed by the reference point and the non-dominated vector vec_i [10].

2.11 Summary

In this chapter evolutionary algorithms basis were described, including main concepts of widely known Genetic Algorithms and Differential Evolution, see Sections 2.2 and 2.3. In Section 2.4, formal concepts of multiobjective optimization was specified as well as differences between classical and evolutionary methods. In Section 2.5, different multiobjective evolutionary algorithms such as NSGA-II and MOEA/D were addressed. This research is focused on solving constrained multiobjective problems, Section 2.6 describes different classical constraint-handling techniques as well as new proposals such PPS and IE. In Sections 2.7, 2.8 and, 2.9 theoretical basis for parallel evolutionary algorithms are specified with emphasis on distributed and cellular models. Finally, in Section 2.10 a number of metrics to assess evolutionary multiobjective algorithms are described. In the following chapter, state of the art for constrained multiobjective evolutionary algorithms is analyzed in detail.

Chapter 3

State of the art

The proposed evolutionary algorithmic approach to tackle constrained multiobjective problems involves a number of concepts from related fields of study. Thus, the state of the art reviews previously proposed EAs approaches to solve constrained multiobjective optimization problems. After, it discusses previous parallel evolutionary approaches to tackle optimization problems either single objective or multiobjective as well as considering constraints or not. Finally, most closely related work, parallel evolutionary schemes previously proposed to solve constrained multiobjective optimization are analyzed in detail.

3.1 Constrained multiobjective evolutionary algorithms

Constrained MOEAs are a very active topic in the area of evolutionary computing, and the following is a review of the most recent proposals in this area. Beginning with the proposal of Xu et. al. [55]. Which develops a variant of DE algorithm, this proposal called IMDE (Infeasible-guiding Mutation Differential Evolution) adds a mechanism to deal with infeasible areas. It adds different variants of the mutation operator to generate mutant vectors in DE, these are the canonical version DE/ran/1/bin, DE/current-2-best/1/bin, and DE/best/1/bin. IMDE modifies GDE3 algorithm [41] using its structure with different changes as the one already mentioned. It makes use of an external set where the most promising infeasible solutions are saved. These solutions are used together with mutation methods to generate more diversity. IMDE was tested with three different benchmark groups, a group with classic functions such as TNK and OSY, another group called CTP, and a last group called NCTP, all of them are constrained nature. Seven MOEAs were used

to make the comparison: MODE-ECHT [56] , SADE-CD [57], NSGA-III [58], MOEA/DD [59], ARMOEA [60], MOEA/D-IEpsilon [27], and ICMOEA [61]. IMDE shows good performance over the other MOEAS, having acceptable results in most benchmarks. Finally, it concludes that the IMDE algorithm has a higher complexity since it handles an additional population, furthermore, the parameters used are sensitive to the type of problems and need to be adapted.

Fan et. al. [28] proposed a new constraint handling technique called PPS (Push and pull search, explained in Section 2.6.1) in conjunction with MOEA/D-DE developed the MOEA/D-PPS proposal. This algorithm is focused on problems that have large infeasible regions. PPS divides the evolutionary process in two phases, at the beginning a problem is approached without constraints, when solutions are reaching faster a constrained Pareto front, after in a second phase, the algorithm starts using Improved Epsilon as a CHT and target the problem with constraints, once solutions are near to the constrained Pareto front. Changing from phase one to phase two is determined by calculating the change rate during last l generations. It calculates changes in values of ideal and nadir points; when changes are less than an epsilon value, it switches to the next phase. MOEA/D-DE algorithm was used as a basis to adapt this CHT, as well as the use of an external archive for saving the best solutions. Solutions in this archive, are updated in each generation according to their non dominance and their crowding distance. For comparison analysis, six other MOEAs were used: MOEA/D-IEpsilon [27] , MOEA/D-Epsilon [62], MOEA/D-SR [42], MOEA/D-CDP [42], C-MOEA/D [63], and NSGA-II-CDP [21]. These were tested on Large Infeasible Regions (LIRCMOP) benchmark. Results showed that this algorithm outperformed the others in most problems in both IGD and HV metrics. The same author on that paper [27] also proposed a CHT called Improved Epsilon, which is adapted to MOEA/D-DE, and its main focus is to solve CMOPs with large infeasible regions. That proposal is called MOEA/D-IEpsilon and its main feature is to dynamically adjust the epsilon parameter according to population feasibility ratio in any current generation, more details on Section 2.6. MOEA/D-IEpsilon was tested on LIRCMOP benchmark against CMOEA/D, MOEA/D-CDP, MOEA/D-SR and MOEA/D-Epsilon algorithms using two common metrics in the area which are IGD and HV. Tests were also performed on a real-world problem which is to optimize grip parameters of a robotic hand. Results on LIRCMOP benchmark and robot grip problem were significantly better than other MOEAs. Authors point out that MOEA/D-IEpsilon is able to explore feasible and infeasible regions simultaneously. Moreover, using population feasibility ratio dynamically balances exploration between

feasible and infeasible areas, thus maintaining a good balance between them.

In [29], Fan et. al. propose a modification to MOEA/D-DE algorithm attaching a CDP variant as a CHT, see Section 2.6. This modification considers an angle between solutions for decisions making. A similarity function defined in Equation 3.1.1 is used. It inputs two vector solutions, normally those for comparison, also the ideal point in current generation is used. It outputs a value to adapt to CDP rules. Thus, its CHT is called ACDP and consequently, the algorithm is known as MOEA/D-ACDP. For evaluation, LIRCMOP benchmark [27] was used and compared against C-MOEA/D, MOEA/D-CDP, MOEA/D-Epsilon [42], MOEA/D-SR [42], NSGA-II-CDP [21], and SP [64] algorithms, results showed better performance in IGD and HV metrics in comparison to the other proposals.

$$angle(\mathbf{x}^1, \mathbf{x}^2, Z^*) = arcos\left(\frac{(\mathbf{F}(\mathbf{x}^1) - Z^*)^T \cdot (\mathbf{F}(\mathbf{x}^2) - Z^*)}{||\mathbf{F}(\mathbf{x}^1) - Z^*|| \cdot ||\mathbf{F}(\mathbf{x}^2) - Z^*||}\right) \tag{3.1.1}$$

A new algorithm based on DE and an improved epsilon CHT called MODE-SaE was proposed by Yang et. al. [65]. Such approach is based on a self-adaptive epsilon level setting considering maximum and minimum values of individuals constraints violation values. MODE-SaE algorithm is compared against five other modified algorithms to deal with CMOPs such as NSGA-II [21] ,RVEA [66], MOEA/DD [59], GDE3 [41], and CTAEA [67]. Experimental results indicate that MODE-SaE is significantly better than the other five constrained MOEAs in most benchmark problems and a real-world problem with a low feasible ratio. This means that problems have a very small space for feasible solutions. Furthermore, MODE-SaE is not sensitive to a feasible ratio. Its weakness is needing more generations to find feasible solutions for CMOPs where the Pareto front is away from unconstrained PF.

Ning et. al. [68] proposed a parameter-free CHT, called constrained non dominated sorting (CNS) and is adapted to the cMOEA/H algorithm, which is based on MOEA/D-DE [19] and MOEA/D-M2M [69]. CNS works first by ranking the population using non-dominated sorting, then calculates the constraint violation sum per individual. Next, it calculates maximum and minimum sum in the population and proceeds to divide the entire population by its degree of a constraint violation. It creates 10 levels, level 1 represents the solutions that did not violate constraints or with the lowest constraint sum up to level 10. Finally, it proceeds to generate a set where solutions are already ordered according to

their Pareto rank (PR) and constraints ranking (CR), using the Equation 3.1.2.

$$CNR(t_i) = max\{PR(t_i), CR(t_i)\}, t_i \in T, i = 1, 2, \ldots, |T| \tag{3.1.2}$$

where T is the population size. cMOEA/H splits a population into K-size subpopulations. It uses the ideal point Z to guide the search and to assign solutions among subpopulations. To test cMOEA/H-CNS a comparison against cMOEA/H-CD, NSGA-II and MOEA/D with different CHTs, such as CDP, SR, penalty methods and e-constrained was carried out. These algorithms were tested over 10 constrained MOPs from the Congress on Evolutionary Computation 2009, results showed that cMOEA/H surpassed NSGA-II algorithm and is competitive with MOEA/D variants.

3.2 Parallel MOEAs and Constrained Parallel MOEAs

This thesis approaches CMOPs through decentralized or cellular evolutionary techniques. There is a lack of research in cellular MOEAs, thus in this thesis, algorithmic characteristics involved in cellular models are explored to obtain better results in CMOPs. From reviewed literature, master-slave and island models have been more commonly used in comparison to few works reported on decentralized, cellular or fine grained model [8]. This section discusses the most recent related work in both areas.

Arias et al. [70] proposed an effective and efficient parallel Differential Evolution algorithm for multiobjective Optimization called pMODE-LD+SS based on the island model. The serial algorithm on which this approach is based uses DE operators as a searching engine and includes two mechanisms for improving its convergence properties. First, it uses a structure to define N closest solutions, for this purpose the euclidean distance between each solution in the objective functions space is measured. Thus, a subset of non-dominated solutions is used as the initial population. A second mechanism is using Tchebycheff's decomposition method for environmental selection, to find solutions that minimize a set of vectors, this idea is similar in MOEA/D algorithm, but in pMODE-LD+SS a fixed size file is maintained. Also, it is necessary to eliminate worst solutions through a metric given by decomposition. For parallelism, islands model is applied with a pollination scheme and bidirectional migration. Migrants selection is random and replacement policy occurs through environmental selection. For comparison NSGA-II, MOEA/D and MOEA/D-DE algorithms were compared while targeting ZDT and DTLZ benchmarks.

For performance metrics hypervolume and Two Set Coverage (C-Metric), results obtained show superior performance in most problems.

Durillo et al. [71] proposed a cellular algorithm based on MOCell and GDE3, thus CellDE algorithm focused on three-objective problems was introduced. This algorithm uses canonical DE/rand/1/bin version used in GDE3. This algorithm implements and asynchronous updating mechanism, in contrast to MOCell that is synchronous [9]. Therefore, immediately after solutions are evolved, these are evaluated and compared. If offspring dominates a current solution it is replaced. If both solutions are non-dominated, then the worst solution within a neighborhood is replaced. Finally, a new solution is saved in an external archive only if it is non-dominated. A feedback mechanism replaces a random solution in the current population with random solutions from the external archive. CellDE was tested against NSGA-II, SPEA2, GDE3, and MOCell algorithms on DTLZ and WFG benchmark. Two metrics HV and additive epsilon indicator ($I^1_{\epsilon+}$) were used [72]. From results obtained, CellDE algorithm obtains better performance in both metrics having a similar performance to GDE3 in certain problems.

In [73] a novel parallel MOEA which is based on GAs and the island model with heterogeneous nodes is proposed called MRMOGA. This algorithm is characterized by encoding solutions with different resolution per island. This algorithm maintains an external archive to save the best solutions which are then selected at random for migration. For replacement, all solutions are first ranked and then replace worst n solutions, MRMOGA uses CDP as a CHT. Empirical testing on ZDT benchmark against a parallel island-based version of NSGA-II with ring topology, using HV metric for comparison is carried out. Results show higher performance than the parallel NSGA-II version.

In [9] authors proposed a multiobjective cellular algorithm called MOCell, see Section 17) for details. This approach uses an external file to store non-dominated solutions (Pareto front) and a feedback mechanism which randomly replaces existing solutions with solutions stored in the external file im every iteration. Authors assess its performance while targeting two types of problems, with and without constraints and compare against NSGA-II and SPEA2 algorithms. CDP is used as CHT, the same used in the NSGA-II algorithm. Results obtained indicate that MOCell algorithm has a superior performance in terms of convergence and HV.

In [10] a multiobjective cellular Particle Swarm Optimization algorithm (MOCPSO) is applied to the problem of the wellbore trajectory design. Conflicting objectives in this problem are to minimize trajectory length, torque, and energy simultaneously. MOCPSO

algorithm uses neighborhoods that adapt according to the number of iterations, thus changing between different neighborhoods schemes (Von Neumann, Moore, Extended Von Neumann). To measure algorithmic performance, the proposed approach is compared against MOPSO, MOEA/D, and NSGA-II. The problem of designing the wellbore trajectory is constrained, the type of constraints used in this problem is a value limitation of certain variables to a specific range. This type of constraints are not so difficult thus CDP is used as CHT. Results showed MOCPSO as statistically superior when compared to MOPSO, MOEA/D, and NSGA-II with a level of significance of 0.05.

In [11] a modification to NSGA-II algorithm by adding a distributed or island scheme is proposed. Every island is divided into a multicore environment, and through migration Pareto Extreme solutions (a subset of non-dominated solutions in the Pareto front) are exchanged. Those solutions migrate to all islands. Authors used the constrained knapsack problem with two and three objectives and ZDT benchmark. HV metric was determined for evaluation while using CDP as a CHT. This algorithm demonstrated that the scheme used is effective and improves solutions diversity, reducing searching time and increasing precision for real and discrete problems according to the results obtained.

Regarding parallel models, Luna et al [50] conducted a wide literature review that covers from 1993 to 2005 and Nebro et al. [8], which covers from 2008 to 2011. In the reviewed work, parallel multiobjective evolutionary algorithms scientific community focuses on three main parallel models: master-Slave, distributed, and cellular. For its simplicity, the master-slave model appears almost in half of the reported approaches, occupying 43% (from 1993 to 2005) and has shown a small increase with respect to what was reported by [8] which is 45% (from 2008 to 2011). On the other hand, the distributed model in related work reported from 1993 to 2005 was 55% and shown a decrease of 10% in publications according to [8]. Finally, [50] reported that cellular models were approached by 2% of related articles between the years 1993 and 2005, this percentage increased in recent years to 8%.

Table 3.1 shows summarized various algorithms related to Constrained Parallel MOEAs work as well as the more recent work related to Constrained MOEAs.

Table 3.1: Summary of the most important works of the state of the art and their most important characteristics.

Author	Algorithmic approach	CHT	MOEA	Parallel Scheme
Jaimes and Coello Coello, 2005 [73]	MRMOGA: Parallel Evolutionary Multiobjective Optimization using Multiple Resolutions.	CDP	GA	Island
Nebro et al., 2007 [9]	MOCell: A Cellular Genetic Algorithm for Multiobjective Optimization	CDP	GA	Cellular
Durillo et al., 2008 [71]	CellDE: Solving Three-Objective Optimization Problems Using a New Hybrid Cellular Genetic Algorithm	-	DE	Cellular
Arias et al., 2010 [70]	pMODE-LD+SS: An Effective and Efficient Parallel Differential Evolution Algorithm for for Multi-Objective Optimization	-	DE	Island
Zheng et al., 2019 [10]	MOCPSO: Multiobjective cellular particle swarm optimization for wellbore trajectory design	CDP	PSO	Cellular
Ning et al., 2017 [68]	cMOEA/H Constrained multi-objective optimization using constrained non-dominated sorting combined with an improved hybrid multi-objective evolutionary algorithm	CNS	MOEA/D-M2M	-
Sato et al., 2018 [11]	Distributed NSGA-II Sharing Extreme Non-Dominated\ Solutions for Constrained Knapsack Problems	CDP	NSGA-II	Island
Yang et al., 2019 [65]	MODE-SaE: A multi-objective differential evolutionary algorithm for constrained multi-objective optimization foproblems with low feasible ratio	Adaptive IE	DE	-
Fan et al., 2019a [27]	An improved epsilon constraint-handling method in MOEA/D for CMOPs with large infeasible regions	IEpsilon	MOEA/D	-
Fan et al., 2019b [28]	Push and pull search for solving constrained multiobjective optimization problems	PPS+IE	MOEA/D-DE	-
Fan et al., 2019c [29]	MOEA/D with angle-based constrained dominance principle for constrained multiobjective optimization problems	ACDP	MOEA/D-DE	-
Xu et al., 2020 [55]	IMDE: Differential evolution with infeasible-guiding mutation operators for constrained multi-objective optimization	Infeasible-Guiding Mutation	GDE3	-
Garcia-Garcia et al., 2020 [20]	**cMOGA/D: a novel cellular GA based on decomposition to tackle constrained multiobjetive problems**	**PPS+IE**	**MOEA/D+MOCell**	**Cellular**

3.3 Summary

In this chapter the state-of-the-art in constrained MOEAs as well as most relevant parallel MOEAs and constrained parallel MOEAs was discussed. From the literature review presented in this chapter. It is observed that MOEA/D algorithm has served as the basis for a number of proposed algorithms, many of them using differential evolution operator. Adapting different CHTs to MOEA/D has led to powerful algorithms that can successfully solve constrained MOPs. Moreover, in the second half of this chapter, different parallel evolutionary algorithms to tackle MOPs and constrained MOPs were analyzed where MOCell and CellDE stand out. From this analysis, it is concluded that parallel MOEAs have been scarcely applied to constrained MOPs, and there is a opportunity niche for research in combination with new CHTs and different approaches such as Pareto or decomposition. In the following chapter, a new algorithm called cMOGA/D is presented, which is inspired by cellular genetic algorithms and well-established MOEA/D principles.

Chapter 4

Cellular Multiobjective Genetic Algorithm based on Decomposition

This chapter describes all components in the algorithmic proposal based on structural properties in cellular GAs which directly impacts the searching process and basic concepts of the MOEA/D algorithm. A general description of MOEA/D basic concepts was provided in Section 2.5.2.

4.1 cMOGA/D algorithm

The proposed algorithm is inspired by structural properties in cellular GAs to tackle Constrained MOPs joined with core principles of MOEA/D. It is called cMOGA/D for short identification. cMOGA/D inherits MOEA/D's neighboring principle due to its structured nature, however, in cMOGA/D neighboring is local and responds only to the population's topology based on a toroidally connected mesh (see Figure 4.1). cMOGA/D uses Push and Pull Search (PPS) as CHT [29]. Figure 4.2 shows all phases of the evolutionary process of the cMOGA/D algorithm, as well as cMOGA/D, is detailed in Algorithm 13. The following subsections explain the phases of the cMOGA/D algorithm.

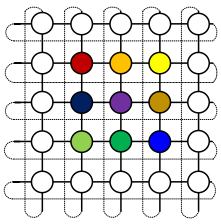


Figure 4.1: Toroidal mesh with a 3×3 neighborhood or Moore.

Algorithm 13: cMOGA/D

$PushStage = 1$;
$\lambda^i = (\lambda^i_1, \ldots, \lambda^i_m)^T, i = 1, \ldots, N_p$;
$B(i) = \{i_1, \ldots, i_t\}$, where $\lambda^{i_1}, \ldots, \lambda^{i_T}$ are the T closest weight vectors to λ^i;
Calculate z^* and n^* points;
$P = \{\mathbf{x}^1, \ldots, \mathbf{x}^{N_p}\}$;
$archive = \emptyset$;
while $k \leq T_{max}$ **do**
    if $k >= l$ **then**
        Set rk using equation 2.6.3;
    if k<Tc **then**
        if $rk < \rho$ *and* $PushStage == 1$ **then**
            $PushStage == 0$;
            $\epsilon(k-1) = maxViolation$;
        if $PushStage == 0$ **then**
            Set $\epsilon(k)$ using equation 2.6.6;
    else
        $\epsilon(k)$=0;
    $\omega = \emptyset$;
    $Front = NDS(P), Crow = Crow(P)$;
    for $i = 1$ **to** $size(P)$ **do**
        $N = Neighbor(i)$;
        $SP = Tournament(Front(P(N)), Crow(P(N)))$;
        Generate y from $\mathbf{x}^{\mathbf{SP(1)}}$ and $\mathbf{x}^{\mathbf{SP(2)}}$ by GA operator;
        Polynomial mutation on y to new solution y^i;
        Set $maxViolation$ using equation 2.6.7;
        $gY = g^{te}(y^i|\lambda^{B(i)}, z^*)$;
        $gN = g^{te}(N|\lambda^{B(i)}, z^*)$;
        if *PushStage==1* **then**
            $\alpha = UpdateSolution(y^i, N, gY, gN, i)$;
            $\omega = \omega \cup \{\alpha\}$;
        else
            $\alpha = ImproveEpsilon(y^i, P, N, gY, gN, \epsilon(k), i)$;
            $\omega = \omega \cup \{\alpha\}$;
        if *Type Update==LS||AS* **then**
            Update Population using ω;
    if *Type Update==SY* **then**
        Update Population using ω;
    $archive = archive \cup y$;
    Update external archive;
    Feedback Mechanics;
    Update z^* and n^* points;

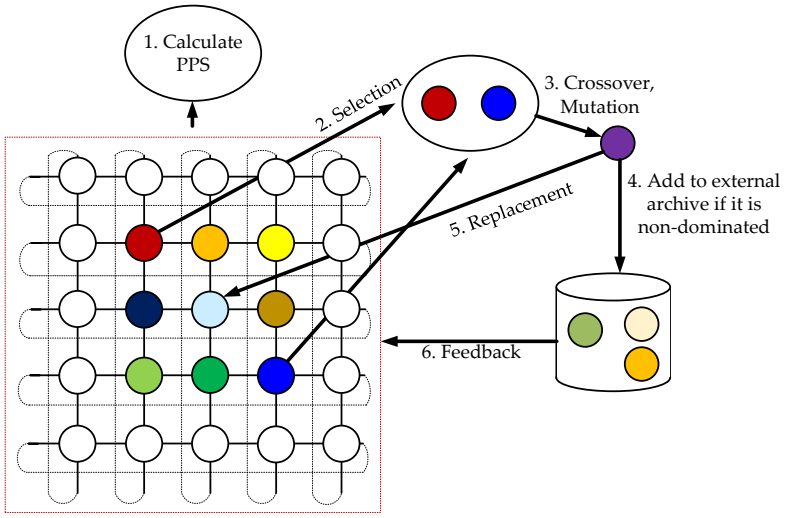


Figure 4.2: Evolutionary process of cMOGA/D algorithm.

4.1.1 Initialization

Cellular MOGA/D needs certain important variables such as λ reference vectors and to generate a structure $B(i)$ where T vectors closest to λ^i vector are stored. cMOGA/D uses PPS and IE as a CHTs, see Section 2.6 for details. Initialization process is written in lines 1 to 6.

cMOGA/D always starts by approaching a CMOP as an unconstrained problem. After in every generation, nadir and ideal points are calculated to switch to stage two where problem's constraints are taken into account. In this stage epsilon ($\epsilon(k)$) value is calculated. This value is used after at replacement stage, all this process is observed in lines 8 to 17 of the Algorithms 13.

4.1.2 Selection, Recombination and Mutation

cMOGA/D algorithm uses a binary tournament for local solutions selection within neighborhoods determined by a toroidally connected mesh. Local Moore neighborhood (square shape) has been deployed, see Figure 4.1. It is formed by 9 solutions directly connected to any current solution.

Binary tournament uses Non-dominated sorting (NDS) and Crowding distance as metrics (lines 19 and 22). NDS specifies a dominance level to all solutions, according to this it defines which solution is better. Moreover, larger crowding distances imply those so-

lution have more chances for selection. Both NDS and crowding distance are calculated in every generation, outside the reproductive cycle. Two best solutions are selected, and they undergo recombination and mutation operations.

Those locally selected solutions are recombined by Simulated Binary Crossing (SBX), and offspring are finally mutated by polynomial mutation, see Sections 2.1.3 and 2.1.4. Lines 21 to 24 in Algorithm 13 show this operation.

4.1.3 Replacement

In cMOGA/D, lambda reference vectors are used to obtain each individual aggregation value and are calculated within neighborhoods using Tchebycheff decomposition (Equation 2.5.1). Initial stage in Algorithm 13 approaches a constraint MOP as an unconstrained problem; this is calculated by PPS as CHT. Thus, solutions within neighborhoods are evaluated in function $UpdateSolution$, see Algorithm 14. According to their aggregation values, these are calculated in lines 26 and 27 of Algorithm 13.

In Algorithm 14, gY is the offspring aggregation vector y^i, and gN is the neighborhood aggregation vector N with respect to λ vectors. j indexes central individuals within neighborhoods and compares aggregation values; if the offspring value is smaller than central's one, this offspring is stored in a temporary vector.

Algorithm 14: $UpdateSolution(y^i, N, gY, gN, i)$

```
1 j = central individual within N neighbourhood;
2 r = ∅;
3 if gY(j) ≤ gN(j) then
4     r = {y^i, N^j}
  Result: r
```

For the second stage, cMOGA/D uses Improved Epsilon as a CHT [27]. Improved Epsilon CHT defines a flexible ϵ parameter which is dynamically reduced as generations elapse until reaching a zero value. It allows a looseness value of the total sum of constraints violation for considering offspring as feasible solutions or not. This dynamic ϵ is updated every generation according to Equation 2.6.6 and its value decreases to zero as it is considered that, in final generations, solutions had reached feasible regions within the landscape. In new versions of MOEA/D, such as MOEA/D-DE [19], nr parameter defines the number of replacements within a neighborhood; however, in the cMOGA/D algorithm, no multiple replacements are applied, so there is a reduction in the comparisons

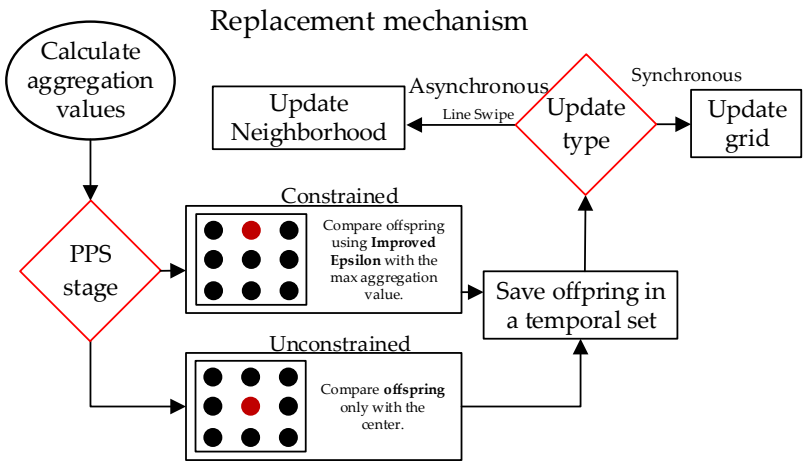


Figure 4.3: Replacement mechanism.

made. The Improved Epsilon algorithm, see Algorithm 15, uses overall constraints violation sum ($\phi()$) for solutions comparison. In this case, offspring and solution for j index, in the first two conditions of the Algorithm 15. Overall constraints violation sum and epsilon value are compared, based on these conditions an extra comparison is made between aggregation values of each solution. If offspring has lower aggregation value this is saved to replace the solution; in the same way, in a second condition aggregation value is taken as a priority. Finally, if neither the first two conditions are satisfied, solution feasibility is taken as a priority. If the solution has a lower overall constraints violation sum, it is chosen to replace the neighborhood solution. Normally, in MOEA/D-DE variants [58], j variable is chosen at random; in cMOGA/D, the index with the highest aggregation value is used to ensure those offspring remain in the next generation. Figure 4.3 shows in detail internal functioning of the replacement mechanism, and also shows how Algorithms 14 and 15 take different paths depending on PPS.

The use of a cellular structure brings the benefit of using different approaches to update the grid; it is at this point that solutions are replaced, here all the solutions stored in ω are used. In cMOGA/D three types of updating were defined: Synchronous, Line sweep, and Asynchronous. Depending on the update type, the algorithm will have different behavior, increasing or decreasing the spread of the solutions in the grid. In Algorithm 11, synchronous update type is taken out of the reproductive cycle, while asynchronous and line sweep are taken into the cycle; this difference makes asynchronous and line sweep

Algorithm 15: $ImprovedEpsilon(y^i, P, N, gY, gN, \epsilon(k), i)$

```
1  r = ∅;
2  j = max(gN);
3  if φ(y^i) ≤ ε(k) and φ(P(N^j)) ≤ ε(k) then
4  |   if gY(j) ≤ gN(j) then
5  |   |   r = {y^i, N^j}
6  else if φ(y^i) == φ(P(N^j)) then
7  |   if gY(j) ≤ gN(j) then
8  |   |   r = {y^i, N^j}
9  else if φ(y^i) < φ(P(N^j)) then
10 |   r = {y^i, N^j}
   Result: r
```

versions more aggressive at introducing new solutions, unlike synchronous version; In Section 2.9.1, details on different updating criteria are provided. Algorithms 10, 11 and 12 are the same ones used by cMOGA/D in lines 35 and 37 of Algorithm 13.

4.1.4 Feedback mechanism

Lines 39 and 40 in Algorithm 13 define an updating mechanism through an external archive. Thus, an external archive is used to store the best solutions in every generation to feedback the population as a good diversity source. This mechanism is inherited from the MOCell algorithm [9] and corresponding procedures are shown in Algorithms 16 and 17.

Algorithm 16: $UpdateExternalArchive(archive, P)$

```
1  archive = NDS(archive, 1);
2  if size(Archive) > size(P) then
3  |   Use crowding distance to rank the archive.
   Result: archive
```

Algorithm 16 shows how an external archive is filled, only offspring evolved in every generation are used and they are filtered by using non-dominated sorting to obtain those at the first front. If an archive exceeds the population's size, crowding distance is used to truncate the archive removing solutions with worse distance. This archive is later used (see Algorithm 17) as the population's feedback considering a maximum δ value for re-

placements. *nRep* is the number of replacements chosen at random. These solutions are placed back aiming at improving diversity and therefore the searching process.

Algorithm 17: $Feedbak(archive, P)$

```
nRep = min(min(δ, size(P)), size(archive));
for i=1 to nRep do
    a, b = rand;
    P(a) = archive(b);
```

Result: P

4.2 Experimental Design

For a thorough empirical assessment, the proposed cMOGA/D is compared against two popular algorithmic approaches: MOEA/D and MOCell using the Push and Pull Search as CHT. For experimental deployment, a multiobjective optimization platform called PlatEmo, [74] has been used. It is fully developed in MatLab and provides several different algorithms and benchmark functions as well as a friendly testing environment with widely accepted performance metrics in the area.

Following parameters configured for each algorithmic approach including the proposed cMOGA/D are shown in Table 4.1. MOEA/D-PPS parameters are the same Authors used in their proposal [28]. cMOGA/D and MOCellPPS use default parameters of most MOEAs that use the genetic operator. The only change is in the ρ variable, this variable decides switching between stage one and two for PPS. This parameter is adjusted since cMOGA/D takes more generations to make this change. Moreover, δ parameter is set to 30 since it is 10% of the population.

4.2.1 Benchmark functions

For empirical validation, a benchmark proposed in [75] is tackled. It has fourteen constrained functions classified in 4 different types:

- Type I: Constrained and unconstrained PF are the same.
- Type II: Constrained Pareto front is a part of unconstrained PF since constraints make a portion of unconstrained PF infeasible.

Table 4.1: Parameters configuration for MOEA/D-PPS, MOCELLPPS and cMOGA/D.

<table>
<tr><th>MOEA/D-PPS</th><th>cMOGA/D</th><th>MOCELLPPS</th></tr>
<tr><td colspan="3">Population size $N = 300$</td></tr>
<tr><td colspan="3">Population $pm = \frac{1}{n}$, where n is the decision vector dimension
Distribution index is set to 20.</td></tr>
<tr><td colspan="2">DE parameters
$CR = 1.0, F = 0.5$</td><td>GA parameters
Crossover ratio $Pc = 1$,
Distribution index for SBX $Dsc = 20$.</td></tr>
<tr><td>Neighborhood size T=30</td><td colspan="2">Neighborhood size T=9</td></tr>
<tr><td>CHT parameters
$T_c = 400, \alpha = 0.95,$
$\tau = 0.1, cp = 2, l = 20,$
$nr = 2, \rho = 1e-3$</td><td>CHT parameters
$T_c = 400, \alpha = 0.95,$
$\tau = 0.1, l = 20,$
$\rho = 1e-2, \delta = 30$</td><td>CHT parameters
$T_c = 800, l = 20, \rho = 1e-3, \delta = 20$</td></tr>
<tr><td colspan="3">Stopping condition: each algorithm is executed 30 independent runs
and stops when 150,000 function evaluations are reached.</td></tr>
</table>

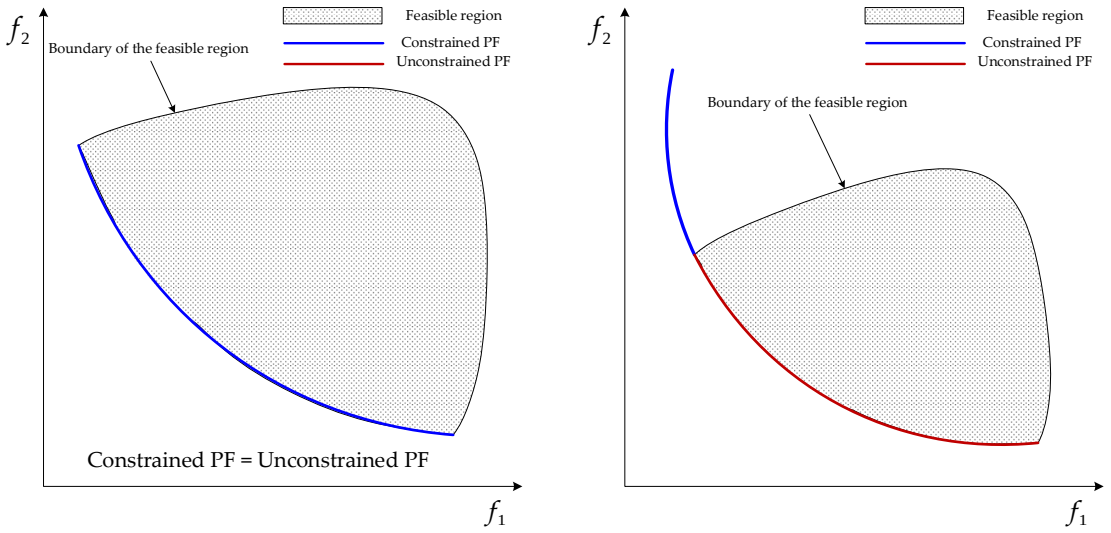
- Type III: Constrained PF is a part of unconstrained PF and a part of a feasible region boundary.

- Type IV: Unconstrained PF is entirely located outside the feasible region. Thus, constrained PF is composed of a part of the feasible region boundary.

This benchmark proposes a more general scheme in constrained multiobjective problems because it offers several features extracted from real-world problems (Figure 4.4 graphically shows the characteristics of the 4 types of problems defined in the MW benchmark). Each type of problem specified above is intended to generalize some real-world problems since if the algorithms were tested on specific problems would require prior knowledge. The authors decided to extract the characteristics of several studies conducted on real-world problems: Low feasibility ratios, sufficient constraints non linearity, more than two constraints, high dimensional decision vectors, convergence difficulty, and diverse geometric PFs [75].

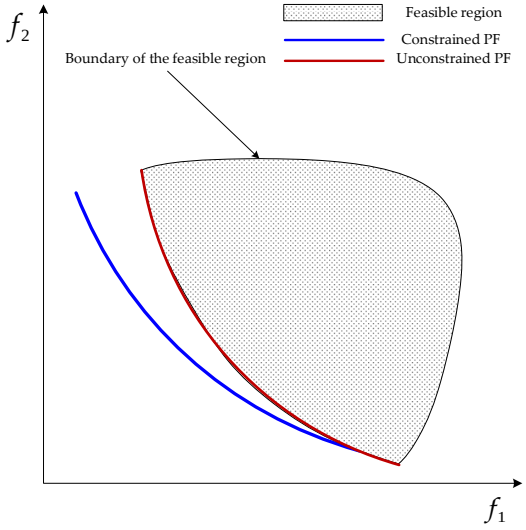
4.3 Results Analysis

Tables 4.2 and 4.3 show overall performance results in terms of Inverted Generational Distance (IGD) and Hypervolume (HV) metrics, these are drawn as average and standard

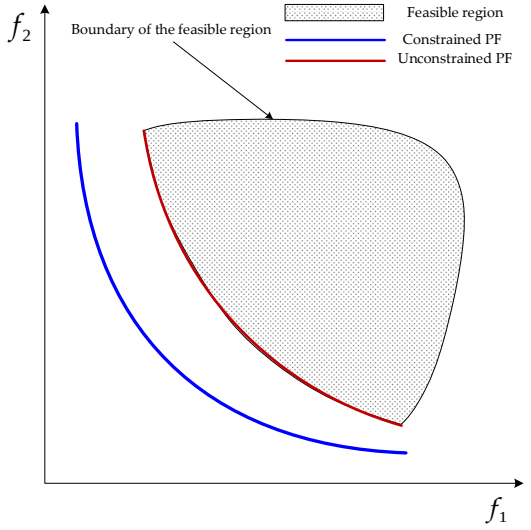


(a) Type I

(b) Type II



(c) Type III



(d) Type IV

Figure 4.4: Constrained multiobjective problems classification of the MW benchmark.

deviation (in parentheses). The proposed approach is empirically compared to two previously proposed approaches known as MOCell and MOEA/D-PPS. cMOGA/D is configured to execute three updating mechanisms: synchronous (SY), line Swipe (LS), and asynchronous (AS). Moreover, Wilcoxon statistical test considering cMOGA/D-SY, its Synchronous version, as reference. In Table 4.2, cMOGA/D shows a significantly better performance to previously proposed approaches MOEA/D-PPS and MOCellPPS in 12 out of 14 problems regarding IGD metric. On the other hand, HV metric in Table 4.3, shows similar results were achieved by cMOGA/D also in 12 out of 14 problems. Both IGD and HV tables show that cMOGA/D-SY, its synchronous version, achieved significantly better results.

Regarding updating mechanisms in cMOGA/D: synchronous, line sweep and asynchronous, results demonstrate that synchronous updating performs better, followed by line swipe and finally asynchronous updating. It can be observed in both tables that cMOGA/D in its synchronous version has difficulties to solve type III problems. In those problems, the constrained Pareto front is part of the unconstrained Pareto front. It is appreciated that algorithmic approaches with an asynchronous updating mechanism, including the well established MOEA/D, achieve better solutions in these type III problems. Moreover, problem's types I, II, and IV are successfully tackled by cMOGA/D-SY, and good results are in general achieved. For an overall perspective of cMOGA/D performance, Figure 4.5 shows obtained results in terms of the average final PFs achieved by the proposed and previous algorithmic approaches for the 14 problems, each one belongs to one problem type mentioned before in Subsection 4.2.1, it can be observed that in most of the Figures 4.5 cMOGA/D-SY (in red) algorithm obtains a better distribution than MOCellPPS (in green) and MOEA/D-PPS (in blue), the difference is more notable against MOCellPPS algorithm, on the other hand, in some Pareto fronts there is almost no difference between cMOGA/D-SY and MOEA/D-PPS, taking as reference MW7 (Figure 4.5g) problem, in which MOEA/D-PPS algorithm according to HV metric is superior to cMOGA/D-SY, it can be seen that difference is very small. Another similar case is MW3 (Figure 4.5c) problem in which according to IGD metric MOEA/D-PPS algorithm is superior. Likewise difference between fronts is minimal, when targeting two objectives problems; in problems MW4, MW8, MW14 (Figures 4.5d, 4.5h, 4.5n), Pareto fronts are in three dimensions, clearly it is more difficult to analyze them because there is not a single general view of the front. However, according to IGD and HV metrics cMOGA/D-SY algorithm is superior to the others, and this is clearly visible in the figures already mentioned, where

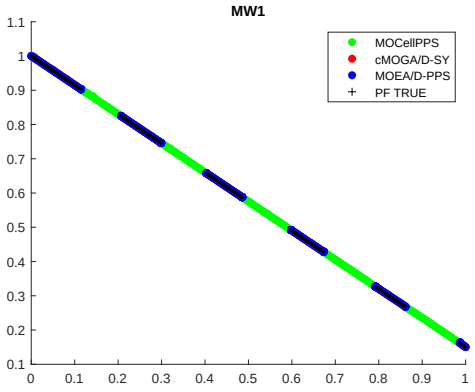
cMOGA/D-SY establishes solutions within feasible zones.

Table 4.2: IGD metric results for cMOGA/D vs MOCell and MOEA/D-PPS on MW benchmark functions. cMOGA/D updating mechanisms: synchronous (SY), line Swipe (LS) and asynchronous (AS). Columns M and D correspond to the number of objectives and vector's dimension, respectively. Wilcoxon statistical test with cMOGA/D-SY as reference is applied, the results marked in grey, are the best results obtained from each function, "+/−/≈" means corresponding algorithm is significantly better, worst or not significantly different to cMOGA/D-SY.

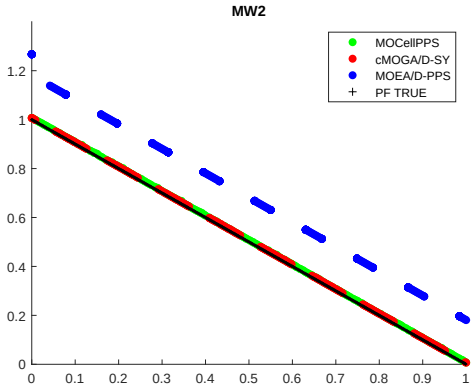
Problem	Type	M	D	MOEA/D-PPS	MOCellPPS	cMOGA/D-LS	cMOGA/D-AS	cMOGA/D-SY
MW1	II	2	15	1.7255e-3 (2.22e-3) −	1.6580e-3 (1.01e-3) −	7.8363e-4 (7.19e-5) ≈	7.9553e-4 (6.67e-5) ≈	7.6049e-4 (5.86e-5)
MW2	I	2	15	1.3095e-1 (5.79e-2) −	1.4683e-2 (5.92e-3) −	1.4040e-2 (6.16e-3) −	1.0074e-2 (5.86e-3) −	7.9668e-3 (4.98e-3)
MW3	III	2	15	2.4055e-3 (3.11e-4) +	4.8579e-2 (8.16e-4) −	3.0580e-3 (4.26e-4) ≈	3.1124e-3 (3.20e-4) −	2.8842e-3 (3.37e-4)
MW4	I	3	15	4.4447e-2 (2.85e-2) ≈	4.4850e-2 (1.50e-3) −	3.7178e-2 (1.34e-3) ≈	3.6996e-2 (1.12e-3) ≈	3.6800e-2 (1.28e-3)
MW5	II	2	15	3.3501e-2 (1.34e-1) −	3.7009e-1 (1.68e-1) −	2.1140e-3 (2.47e-3) ≈	1.9583e-3 (1.68e-3) ≈	1.8314e-3 (2.22e-3)
MW6	II	2	15	5.0829e-1 (2.49e-1) −	1.1672e-2 (7.70e-3) −	4.4656e-3 (4.58e-3) ≈	8.0826e-3 (6.62e-3) ≈	6.9728e-3 (5.67e-3)
MW7	III	2	15	3.0890e-3 (3.33e-4) −	2.0169e-1 (8.16e-4) −	2.6288e-3 (3.04e-4) ≈	2.6708e-3 (2.74e-4) ≈	2.7359e-3 (3.65e-4)
MW8	II	3	15	1.3458e-1 (4.78e-2) −	4.0319e-2 (3.18e-3) −	3.2238e-2 (2.41e-3) ≈	3.2734e-2 (3.54e-3) ≈	3.2754e-2 (2.95e-3)
MW9	IV	2	15	4.6264e-1 (6.00e-1) −	6.3139e-1 (1.40e-4) −	2.9315e-3 (2.20e-4) −	2.9368e-3 (2.19e-4) ≈	2.8337e-3 (2.15e-4)
MW10	III	2	15	3.9197e-1 (1.98e-1) −	3.0459e-2 (2.19e-2) −	9.8878e-3 (9.73e-3) ≈	7.8977e-3 (7.48e-3) ≈	8.4577e-3 (1.06e-2)
MW11	IV	2	15	5.3170e-3 (3.59e-4) −	NaN (NaN)	3.3982e-3 (4.23e-4) −	3.7239e-3 (6.07e-4) −	2.9818e-3 (1.96e-4)
MW12	IV	2	15	9.0947e-2 (1.90e-1) −	NaN (NaN)	2.7820e-2 (1.39e-1) ≈	2.5434e-3 (1.43e-4) −	2.4455e-3 (1.41e-4)
MW13	III	2	15	3.4373e-1 (2.44e-1) −	1.4007e-1 (3.10e-2) −	2.6674e-2 (2.49e-2) ≈	3.0510e-2 (2.48e-2) −	1.6437e-2 (1.39e-2)
MW14	I	3	15	1.7968e-1 (1.63e-2) −	2.1555e-1 (1.15e-1) −	1.1787e-1 (8.23e-2) ≈	1.1417e-1 (6.51e-2) ≈	1.0211e-1 (5.01e-2)
+/−/≈				1/12/1	0/12/0	0/3/11	0/5/9	

Table 4.3: HV metric results for cMOGA/D vs MOcell and MOEA/D-PPS on MW benchmark functions. cMOGA/D updating mechanisms: synchronous (SY), line Swipe (LS) and asynchronous (AS). Columns M and D correspond to the number of objectives and vector's dimension, respectively. Wilcoxon statistical test with cMOGA/D-SY as reference is applied, the results marked in grey are the best results obtained from each function, "+/−/≈" means corresponding algorithm is significantly better, worst or not significantly different to cMOGA/D-SY.

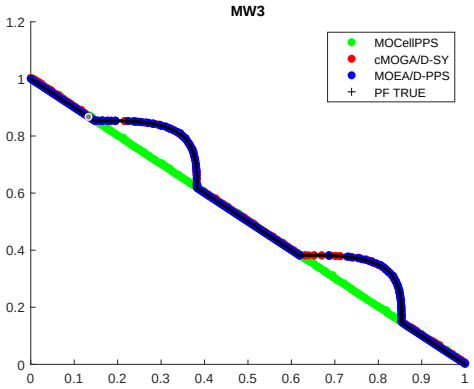
Problem	Type	M	D	MOEA/D-PPS	MOCellPPS	cMOGA/D-LS	cMOGA/D-AS	cMOGA/D-SY
MW1	II	2	15	4.8846e-1 (4.45e-3) −	4.8803e-1 (1.98e-3) −	4.9000e-1 (1.81e-4) ≈	4.8994e-1 (1.94e-4) −	4.9008e-1 (1.49e-4)
MW2	I	2	15	4.0965e-1 (6.82e-2) −	5.6379e-1 (1.06e-2) −	5.6477e-1 (1.05e-2) −	5.7162e-1 (9.89e-3) −	5.7510e-1 (8.29e-3)
MW3	III	2	15	5.4788e-1 (6.96e-5) +	5.3642e-1 (7.73e-4) −	5.4554e-1 (7.40e-4) −	5.4549e-1 (6.92e-4) −	5.4587e-1 (6.53e-4)
MW4	I	3	15	8.1742e-1 (4.44e-2) −	8.3252e-1 (1.87e-3) −	8.3994e-1 (2.18e-3) ≈	8.3971e-1 (1.86e-3) ≈	8.4023e-1 (1.90e-3)
MW5	II	2	15	3.1117e-1 (4.17e-2) −	2.0176e-1 (3.72e-2) −	3.2369e-1 (8.96e-4) ≈	3.2376e-1 (6.12e-4) ≈	3.2367e-1 (1.51e-3)
MW6	II	2	15	1.0947e-1 (6.40e-2) −	3.1269e-1 (1.05e-2) −	3.2419e-1 (6.55e-3) ≈	3.1910e-1 (9.17e-3) ≈	3.2062e-1 (7.94e-3)
MW7	III	2	15	4.1367e-1 (1.69e-4) +	3.4376e-1 (8.16e-4) −	4.1331e-1 (4.15e-4) ≈	4.1320e-1 (3.23e-4) ≈	4.1322e-1 (5.72e-4)
MW8	II	3	15	3.5429e-1 (7.61e-2) −	5.3260e-1 (8.56e-3) −	5.4476e-1 (7.98e-3) ≈	5.4328e-1 (1.04e-2) ≈	5.4343e-1 (1.02e-2)
MW9	IV	2	15	1.9664e-1 (1.66e-1) −	9.0533e-2 (2.70e-4) −	4.0137e-1 (1.32e-3) ≈	4.0130e-1 (1.44e-3) ≈	4.0117e-1 (1.15e-3)
MW10	III	2	15	2.2899e-1 (9.06e-2) −	4.2707e-1 (1.95e-2) −	4.4471e-1 (1.15e-2) ≈	4.4686e-1 (9.61e-3) ≈	4.4663e-1 (1.17e-2)
MW11	IV	2	15	4.4752e-1 (2.15e-4) −	NaN (NaN)	4.4753e-1 (3.54e-4) −	4.4670e-1 (8.87e-4) −	4.4807e-1 (3.16e-4)
MW12	IV	2	15	5.2279e-1 (1.67e-1) −	NaN (NaN)	5.8643e-1 (1.11e-1) ≈	6.0653e-1 (2.67e-4) ≈	6.0670e-1 (2.77e-4)
MW13	III	2	15	2.9622e-1 (9.32e-2) −	4.3635e-1 (1.57e-2) −	4.6616e-1 (1.14e-2) ≈	4.6339e-1 (1.07e-2) −	4.7000e-1 (7.54e-3)
MW14	I	3	15	4.4427e-1 (6.44e-3) −	4.1241e-1 (4.95e-2) −	4.5689e-1 (2.31e-2) ≈	4.5714e-1 (2.03e-2) ≈	4.6195e-1 (1.40e-2)
+/−/≈				2/12/0	0/12/0	0/3/11	0/5/9	


MW1
MOCellPPS
cMOGA/D-SY
MOEA/D-PPS
PF TRUE

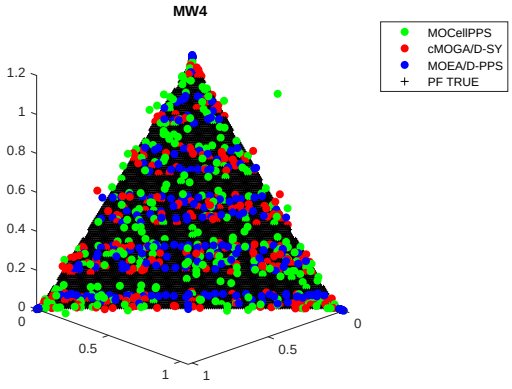
(a)


MW2
MOCellPPS
cMOGA/D-SY
MOEA/D-PPS
PF TRUE

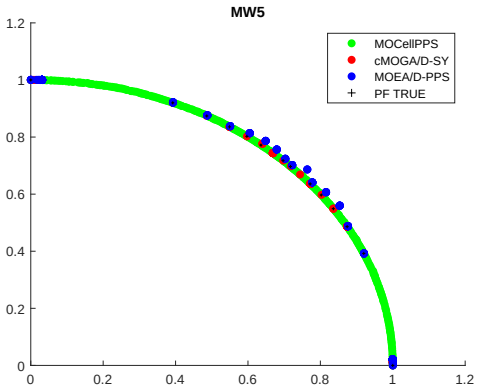
(b)


MW3
MOCellPPS
cMOGA/D-SY
MOEA/D-PPS
PF TRUE

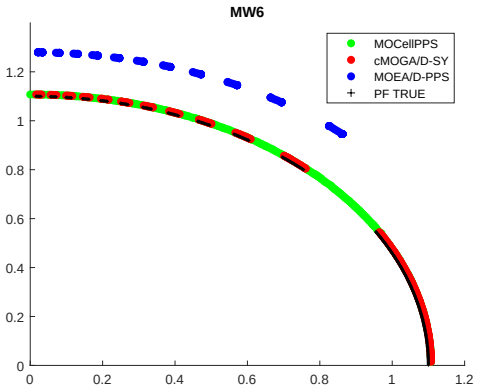
(c)


MW4
MOCellPPS
cMOGA/D-SY
MOEA/D-PPS
PF TRUE

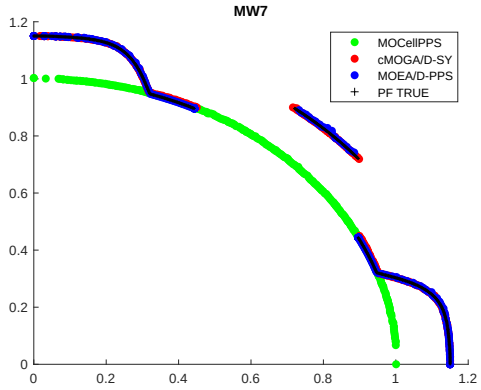
(d)


MW5
MOCellPPS
cMOGA/D-SY
MOEA/D-PPS
PF TRUE

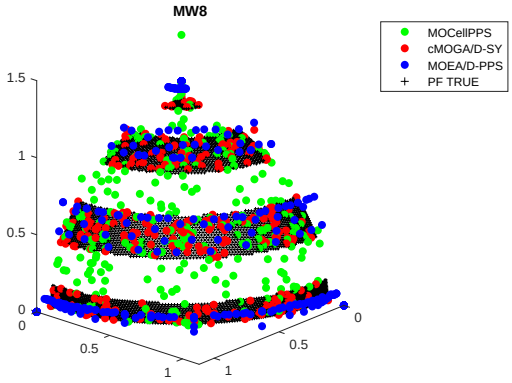
(e)


MW6
MOCellPPS
cMOGA/D-SY
MOEA/D-PPS
PF TRUE

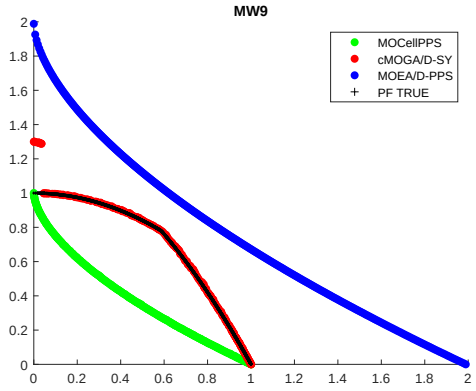
(f)


MW7
MOCellPPS
cMOGA/D-SY
MOEA/D-PPS
PF TRUE

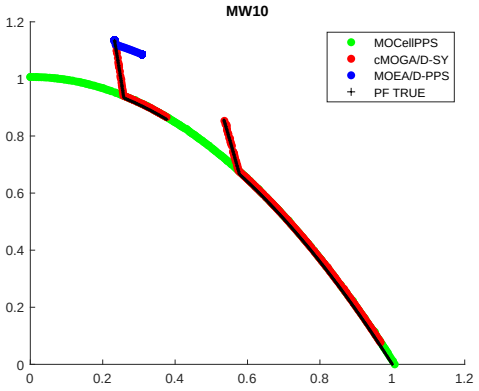
(g)


MW8
MOCellPPS
cMOGA/D-SY
MOEA/D-PPS
PF TRUE

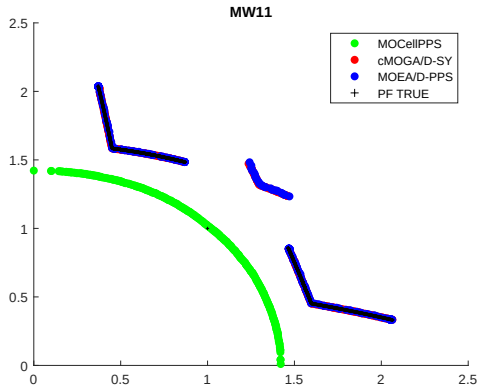
(h)


MW9
MOCellPPS
cMOGA/D-SY
MOEA/D-PPS
PF TRUE

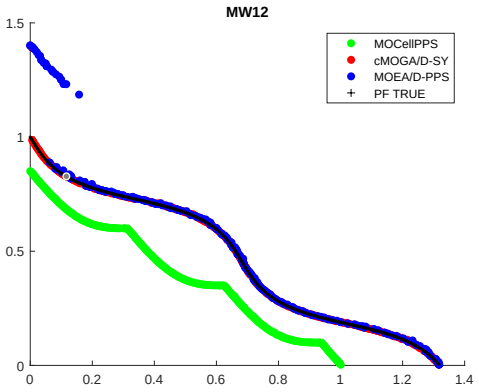
(i)


MW10
MOCellPPS
cMOGA/D-SY
MOEA/D-PPS
PF TRUE

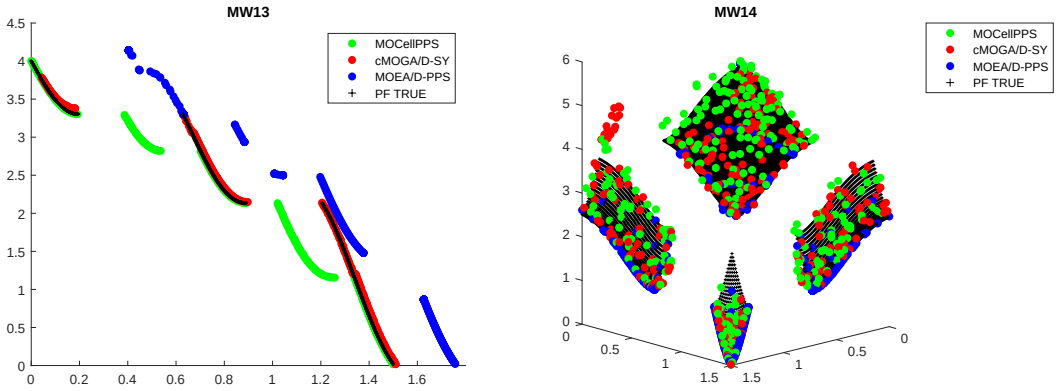
(j)


MW11
MOCellPPS
cMOGA/D-SY
MOEA/D-PPS
PF TRUE

(k)


MW12
MOCellPPS
cMOGA/D-SY
MOEA/D-PPS
PF TRUE

(l)



(m) (n)

Figure 4.5: Average PFs for MOCell, MOEA/D-PPS and the proposed algorithm cMOGA/D-SY on the 14 test problems of MW benchmark.

4.3.1 Time Analysis

Experimental assessment was carried out on a processor Intel Core I5 2.30 GHz, in 64 bits Windows 10 operating system, with 8GB RAM. Code was written in MATLAB R2019a, all tests were performed on the PlatEMO platform [74] fully developed in MATLAB [76]. For time analysis, the total time for the algorithm to run was calculated. After the algorithm was executed 30 times independently. In the same way as IGD and HV metrics; Table 4.4 shows the average in seconds that an algorithm takes to complete the 150,000 assigned evaluations for every problem.

Results show that MOEA/D-PPS had the slowest processing times, this is due to the size of the neighborhoods used by MOEA/D-PPS which are proportional to the population's size. Therefore, as the population size increases, neighborhoods tend to become larger. It was mentioned in previous sections that MOEA/D-PPS neighborhoods are based on reference points that are uniformly distributed in the solution space. MOEA/D-PPS takes longer to calculate and perform a solution replacement; because in first instance MOEA/D-PPS takes random indexes to compare between offspring and parents. Later this process is made *nr* number of times until any of the two conditions are satisfied. First condition if all solutions are evaluated against offspring, if any of them is worst there is not replacement. On contrary, if offspring is better, this process is carried out *nr* number of times until the maximum number of replacements is fulfilled, making the algorithm computationally more expensive. Unlike MOEA/D-PPS, neighborhoods used in cellular algorithms are not related to population size; therefore, in cMOGA/D and MOCellPPS with Moore neighborhoods, the number of comparisons is reduced. Moreover, in cMOGA/D, there are not multiple replacements, this is reflected in Table 4.4 with shorter times than MOEA/D-PPS.

It can be noted that MOCellPPS algorithm takes less time to evaluate than the other algorithms; this is because MOCellPPS algorithm is based entirely on non-dominance and all comparisons are made using the hierarchy given by that scheme. cMOGA/D and MOEA/D-PPS use non-dominance as well as the decomposition approach. This adds more complexity to the algorithm because it takes more steps to calculate it. Nevertheless, the difference that exists in terms of time between both algorithms is small. Considering IGD and HV metrics for overall performance, cMOGA/D achieves much better performance both in time and algorithmic performance metrics obtained.

Table 4.4: Runtime results for cMOGA/D vs MOCell and MOEA/D-PPS on MW benchmark functions. cMOGA/D updating mechanisms: synchronous (SY), line Swipe (LS) and asynchronous (AS). Columns M and D correspond to the number of objectives and vector's dimension, respectively. Wilcoxon statistical test with cMOGA/D-SY as reference is applied, the results marked in grey are the best results obtained from each function, "+/−/≈" means corresponding algorithm is significantly better, worst or not significantly different to cMOGA/D-SY.

Problem	M	D	MOEA/D-PPS	MOCellPPS	cMOGA/D-LS	cMOGA/D-AS	cMOGA/D-SY
MW1	2	15	1.8461e+2 (1.77e+1)	6.4010e+1 (5.44e+0) +	6.4575e+1 (4.11e+0) +	5.8516e+1 (4.30e+0) +	7.0383e+1 (5.69e+0) +
MW2	2	15	1.8774e+2 (1.55e+1)	4.9164e+1 (3.67e+0) +	6.1964e+1 (5.10e+0) +	5.3287e+1 (3.78e+0) +	6.4259e+1 (4.54e+0) +
MW3	2	15	1.8493e+2 (1.94e+1)	5.0877e+1 (4.34e+0) +	6.1651e+1 (4.01e+0) +	5.4801e+1 (3.25e+0) +	6.7733e+1 (6.31e+0) +
MW4	3	15	1.4730e+2 (1.28e+1)	6.8132e+1 (7.47e+0) +	6.5284e+1 (4.15e+0) +	5.6738e+1 (3.55e+0) +	6.6713e+1 (4.36e+0) +
MW5	2	15	1.9289e+2 (1.40e+1)	5.8431e+1 (4.23e+0) +	6.7023e+1 (4.10e+0) +	5.8043e+1 (4.97e+0) +	6.5348e+1 (4.48e+0) +
MW6	2	15	1.7618e+2 (1.69e+1)	5.1913e+1 (3.07e+0) +	6.1763e+1 (3.86e+0) +	5.2858e+1 (3.44e+0) +	6.2979e+1 (4.50e+0) +
MW7	2	15	1.5707e+2 (9.99e+0)	5.4559e+1 (4.93e+0) +	6.3900e+1 (3.84e+0) +	5.8170e+1 (3.80e+0) +	6.9309e+1 (4.15e+0) +
MW8	3	15	1.4182e+2 (1.09e+1)	6.5574e+1 (6.41e+0) +	6.5040e+1 (3.92e+0) +	5.6733e+1 (3.70e+0) +	6.6136e+1 (4.21e+0) +
MW9	2	15	1.3689e+2 (9.37e+0)	5.6032e+1 (5.04e+0) +	6.3391e+1 (4.09e+0) +	5.3801e+1 (3.24e+0) +	6.3916e+1 (4.01e+0) +
MW10	2	15	1.7341e+2 (1.89e+1)	4.8288e+1 (3.73e+0) +	6.3023e+1 (4.03e+0) +	5.4540e+1 (3.35e+0) +	6.4364e+1 (4.01e+0) +
MW11	2	15	1.5335e+2 (9.12e+0)	4.8219e+1 (3.67e+0) +	6.7583e+1 (3.75e+0) +	6.1350e+1 (4.35e+0) +	7.1164e+1 (4.34e+0) +
MW12	2	15	1.5889e+2 (9.95e+0)	5.0345e+1 (3.34e+0) +	6.5601e+1 (4.32e+0) +	5.6265e+1 (3.90e+0) +	6.6987e+1 (5.07e+0) +
MW13	2	15	1.3785e+2 (8.67e+0)	4.8626e+1 (4.63e+0) +	7.1252e+1 (6.66e+0) +	5.8563e+1 (4.91e+0) +	6.6167e+1 (4.66e+0) +
MW14	3	15	1.2624e+2 (7.79e+0)	5.4054e+1 (6.20e+0) +	6.9968e+1 (5.16e+0) +	6.0669e+1 (3.89e+0) +	7.1636e+1 (4.80e+0) +
+/−/≈				14/0/0	14/0/0	14/0/0	14/0/0

4.3.2 Algorithm complexity for evolutionary algorithms

For an in-depth analysis, a complexity analysis for evolutionary algorithm that have been used on competitions at the Congress on Evolutionary Computation, one of the top conferences in the area, is carried out [77, 78, 79, 80, 81]. In the evolutionary computing community, the term complexity algorithm was assigned to this procedure although it is not closely related to formal mathematical analysis. The main idea behind this analysis is to define what is the exact time that algorithm takes without counting the evaluation process of objective functions. Finally, this measure is divided by a value that is defined by the time it takes for the processing platform to make a defined number of calculations to normalize the final result. Three variables are involved in this process, they are defined as follows:

- A) $T0$ is obtained by evaluating the machine time to perform the following evaluations:
  $for\ i = 1 : 100000$
  $x = 0.55 + i;$

$x = x + x; x = x/2; x = x * x; x = sqrt(x); x = log(x); x = exp(x); x = x/(x + 2);$
end

- B) $T1$ is obtained by evaluating the time it takes to evaluate a chosen function n number of times. In this case, 150,000 evaluations were used and the MW3 problem of the MW benchmark was chosen.

- (c) $T2$ is the time for the algorithm to solve a function with a maximum number of evaluations, from this value, $\hat{T2}$ value is calculated. $\hat{T2}$ is the average running time for 30 executions targeting the MW3 benchmark problem.

Once these three values are calculated the following calculation is carried out $(\hat{T2} - T1)/T0$. The value obtained from this operation reflects the total time it takes for the algorithm to perform all evolutionary stages (selection, crossover, mutation, and replacement) while removing the time it takes to evaluate the target function, which is normally the bottleneck calculation in most evolutionary optimization algorithms. In Table 4.5, it can be seen the values obtained.

Table 4.5: Results of the operation $(\hat{T2} - T1)/T0$.

Algorithm	$(\hat{T2} - T1)/T0$
MOEA/D-PPS	2.127085858842389e+03
MOCellPPS	4.103302217426470e+02
cMOGA/D-SY	5.973353766180987e+02
cMOGA/D-LS	5.689036554893037e+02
cMOGA/D-AS	4.614082113783865e+02

From Table 4.5, it can be seen that these results effectively confirm results shown in Table 4.4, with MOEA/D-PPS being the slowest-running algorithm. For MOCellPPS algorithm, as explained in Section 4.3.1, since it is based on non-dominance and is simpler than cMOGA/D, it is faster, but its performance is worst in IGD and HV metrics.

On the other hand, it can be noticed how depending on the updating criteria that cMOGA/D uses, it increases the time that it takes to complete all evaluations. Asynchronous updating is the fastest from the three, following Line Sweep and finally Synchronous criterion. This behavior is given by how the grid is updated, synchronous updating keeps in a temporary variable all solutions to replace, and wait to finish the generation for updating. In contrast, asynchronous updating does not use this temporary variable. As

a middle case, Line Sweep criterion uses the temporary variable but a current generation does not need to finish for updating.

4.4 Summary

Combining cellular genetic algorithms and MOEA/D decomposition principles lead the way to cMOGA/D algorithm to tackle constrained MOPs. It uses Push and Pull search and Improved Epsilon as CHTs. cMOGA/D implements three update types: Synchronous, Line Sweep, and Asynchronous, which have different effects on how solutions are spread throughout the mesh. From a thorough empirical assessment on the MW benchmark, it is observed that cMOGA/D in its synchronous version obtains a superior performance in comparison to MOEA/D-PPS and MOCellPPS algorithms on IGD and HV metrics. A time analysis was also conducted, concluding that cMOGA/D is faster than MOEA/D-PPS.

The next chapter extends the proposed algorithmic approach to apply differential evolution operators in combination with decomposition concepts to solve constrained multiobjective problems with large unfeasible areas from LIR-CMOPs benchmark.

Chapter 5

Cellular Multiobjective Differential Evolution Algorithm based on Decomposition

Benchmark problems for constrained optimization present landscapes with challenging characteristics such as large zones of infeasibility, complex-shaped Pareto fronts, as well as concave and convex Pareto fronts. In order to further demonstrate robustness of the proposed approach based on cellular EAs to tackle constrained multiobjective problems. In this chapter, LIRCMOP problems, which present large infeasible regions, are tackled through a cellular evolutionary model while using Differential Evolution operations to drive the search.

5.1 cMODE/D Algorithm

The proposed cMODE/D algorithm uses the differential evolution operator to tackle Large Infeasible Regions-Constrained Multiobjective Problems (LIRCMOP) [27]. The main feature of this benchmark is that constraints generate large infeasible regions, this makes the evolutionary process much more difficult because it must be able to get through these regions to reach the Pareto front. cMODE/D is based on properties of cellular evolutionary algorithms in combination with MOEA/D principles. In this case, differential evolution operators are used, specifically DE/rand/1/bin version, since it has been widely used to deal with constrained problems [43, 29, 28, 27, 68]. cMODE/D is based on a fully connected toroidal grid, cMODE/D uses PPS and IE as CHT. Figure 5.1, shows all phases of the evolutionary process for cMODE/D algorithm, and it is detailed in Algorithm 18.

Algorithm 18: cMODE/D

$PushStage = 1$;
$\lambda^i = (\lambda^i_1, \ldots, \lambda^i_m)^T, i = 1, \ldots, N_p$;
$B(i) = \{i_1, \ldots, i_t\}$, where $\lambda^{i_1}, \ldots, \lambda^{i_T}$ are the T closest weight vectors to λ^i.;
Calculate z^* and n^* points;
$P = \{\mathbf{x}^1, \ldots, \mathbf{x}^{N_p}\}$;
$archive = \emptyset$;
while $k \leq T_{max}$ **do**
  if $k >= l$ **then**
    Set rk using equation 2.6.3;
  $pf = \frac{\phi(x)}{size(P)}$ feasibility rate;
  if $k<Tc$ **then**
    if $rk < \rho$ *and* $PushStage == 1$ **then**
      $PushStage == 0$;
      $\epsilon(k-1) = maxViolation$;
    if $PushStage == 0$ **then**
      Set $\epsilon(k)$ using equation 2.6.6;
  else
    $\epsilon(k)$=0;
  $\omega = \emptyset$;
  $Front = NDS(P), Crow = Crow(P)$;
  for $i = 1$ **to** $size(P)$ **do**
    $N = Neighbor(i)$;
    $SP = Tournament(Front(P(N)), Crow(P(N)))$;
    Generate y from $\mathbf{x}^{\mathbf{SP(1)}}$, $\mathbf{x}^{\mathbf{SP(2)}}$ and $\mathbf{x}^{\mathbf{SP(3)}}$ by DE operator;
    Polynomial mutation on y to new solution y^i;
    Set $maxViolation$ using equation 2.6.7;
    $gY = g^{te}(y^i|\lambda^{B(i)}, z^*)$;
    $gN = g^{te}(N|\lambda^{B(i)}, z^*)$;
    if *PushStage==1* **then**
      $\alpha = UpdateSolution(y^i, P, N, gY, gN, i, nr)$;
      $\omega = \omega \cup \{\alpha\}$;
    else
      $\alpha = ImproveEpsilon(y^i, P, N, gY, gN, \epsilon(k), i, nr)$;
      $\omega = \omega \cup \{\alpha\}$;
    if *Type Update==LS||AS* **then**
      Update Population using ω;
  if *Type Update==SY* **then**
    Update Population using ω;
  $archive = archive \cup y$;
  Update external archive;
  if $pf \leq 0.5$ **then**
    Feedback Mechanics;
  else
    Replace Worst Feedback Mechanics;
  Update z^* and n^* points;

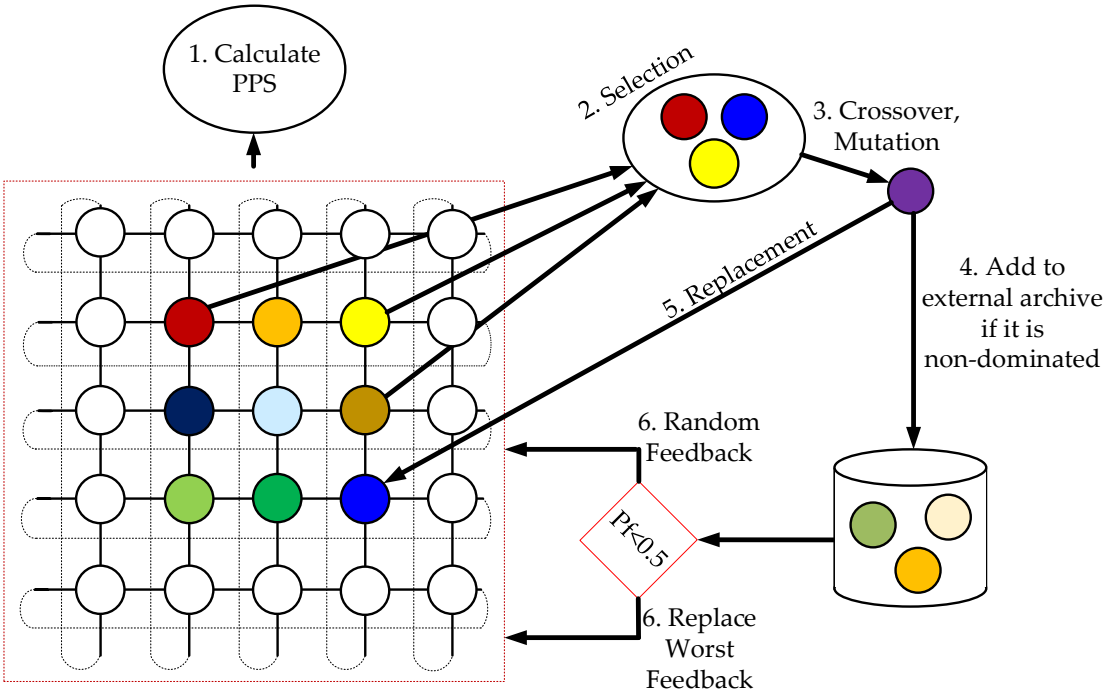


Figure 5.1: Evolutionary process of cMODE/D algorithm, Pf is the population feasibility.

5.1.1 Initialization

Before parents' selection, cMODE/D needs to define certain important variables such as λ reference vectors and to generate a structure $B(i)$ where T vectors closest to λ^i vector are stored. Later on, as already mentioned, cMODE/D uses PPS as a CHT as well as IE (see Section 2.6). Initialization process is observed in lines 1 to 6.

cMODE/D always starts in stage one approaching the problem without constrains, later on in each generation, nadir and ideal points are calculated to switch to stage two where and target the problem with constraints. In stage two, epsilon ($\epsilon(k)$) value is calculated, it is used later in the replacement stage. This process is observed in lines 8 to 18 of Algorithm 18.

5.1.2 Selection, recombination and mutation

cMODE/D algorithm uses a binary tournament for local solutions selection within neighborhoods determined by a toroidally connected mesh such as cMOGA/D, local Moore neighborhood (square shape) has been deployed, it is formed by 9 solutions directly con-

nected to any current solution. Binary tournament uses Non-dominated sorting (NDS) and Crowding distance as metrics (lines 19 and 22 in Algorithm 18). NDS specifies a dominance level to all solutions, depending on it defines which solution is better. A larger crowding distance for a solution means, it has higher probability to be selected. It should be specified that both NDS and crowding distance are calculated every generation, outside the reproductive cycle. Thus, three best solutions are selected, and these will go into the recombination and mutation phase. Those locally selected solutions are recombined by Differential Evolution operator using Equation 2.5.2. Offspring are finally mutated by polynomial mutation, see Sections 2.3,2.5.2 and 2.1.4. Recombination and mutation are detailed in lines 20 to 25 of Algorithm 18.

5.1.3 Replacement

In cMODE/D, lambda reference vectors are used to obtain each individual aggregation value and are calculated within neighborhoods using Tchebycheff decomposition (Equation 2.5.1). The initial stage in Algorithm 18 approaches a constraint MOP as an unconstrained problem; this is calculated by PPS as a contraint handling technique. Thus, solutions within neighborhoods are evaluated in function $UpdateSolution$, see Algorithm 19 and according to their aggregation values. See lines 27 and 28 of Algorithm 18.

Different from cMOGA/D, where only one comparison and one replacement is done, in cMODE/D a stronger impact throughout the search is given by the replacement criterion through an extra parameter nr, it allows increasing the number of comparisons and the maximum number of replacements within the neighborhood.

In Algorithm 19, nr parameter is included as a control to the maximum number of replacements that will be made within the neighborhood. Vectors gY and gN are compared in a random index j, since multiple comparisons are made, any solution in the neighborhood is given the opportunity to be compared to offspring or evolved solutions. This process is repeated until nr number of replacements have been made, or all solutions in the neighborhood have been compared.

For the second stage, cMODE/D uses Improved Epsilon as CHT [27]. Improved Epsilon CHT defines a flexible ϵ parameter which is dynamically reduced with generations. It allows a lossless value of the total sum of constraints violation for considering offspring as feasible solutions or not. This dynamic ϵ is updated every generation according to Equation 2.6.6 and its value decreases to zero as it is considered that, in final generations,

Algorithm 19: $UpdateSolution(y^i, P, N, gY, gN, i, nr)$

```
1 r = ∅, c = 0;
2 while P ≠ ∅ or c ≠ nr do
3     j = uniform random number between 1 and size(P);
4     if gY(j) ≤ gN(j) then
5         r = r ⋃ {y^i, N^j};
6         c = c + 1;
7     p(j), gY(j), gN(j) = ∅;
```

Result: r

solutions had reached feasible regions within the landscape.

The Improved Epsilon algorithm, see Algorithm 20, utilizes overall constraints violation sum ($\phi()$) of solutions for comparison. Similar to Algorithm 19, nr parameter is used, and variable j is chosen as the solution in the neighborhood with the highest aggregation value. The idea is to replace as many bad solutions as possible.

In the first two conditions of the Algorithm 20, the overall constraints violation sum is compared, as well as the epsilon value, depending on these conditions another comparison is made between aggregation values of each solution. If offspring has smaller aggregation value, it is saved to replace the solution; in the same way, in the second condition aggregation value is taken as a priority. Finally, if neither first two conditions are satisfied, in the last condition solution's feasibility is taken as a priority. Thus, if a solution has a lower overall constraints violation sum it is chosen to replace the neighborhood solution.

Figure 5.2 shows in detail internal functioning of the replacement mechanism. This is also shown in Algorithms 19 and 20 considering different algorithmic steps according to PPS.

Using a cellular structure provides the benefit of using different approaches to update the grid. It is at this point that solutions are replaced. All solutions stored in ω are used. Similar to cMOGA/D, in cMODE/D, three types of updating are defined: Synchronous, Line Sweep, and Asynchronous. Depending on updating criterion, the algorithm will have different behavior, increasing or decreasing the spread of the solutions througout the grid. In Algorithm 18, synchronous update type is taken out of the reproductive cycle, while asynchronous and line sweep are taken into the cycle. This difference makes asynchronous and line sweep versions more aggressive at introducing new solutions, unlike the synchronous version. In Section 2.9.1, those different updating criteria are explained in details. Algorithms 10, 11 and 12 are common to cMOGA/D and cMODE/D in lines 35

Algorithm 20: $ImproveEpsilon(y^i, P, N, gY, gN, \epsilon(k), i, nr)$

```
r = ∅, c = 0;
while P ≠ ∅ or c ≠ nr do
    j = max(gN);
    if φ(y^i) ≤ ε(k) and φ(P(N^j)) ≤ ε(k) then
        if gY(j) ≤ gN(j) then
            r = r ⋃ {y^i, N^j};
            c = c + 1;
    else if φ(y^i) == φ(P(N^j)) then
        if gY(j) ≤ gN(j) then
            r = r ⋃ {y^i, N^j};
            c = c + 1;
    else if φ(y^i) < φ(P(N^j)) then
        r = r ⋃ {y^i, N^j};
        c = c + 1;
    p(j), gY(j), gN(j) = ∅;
Result: r
```

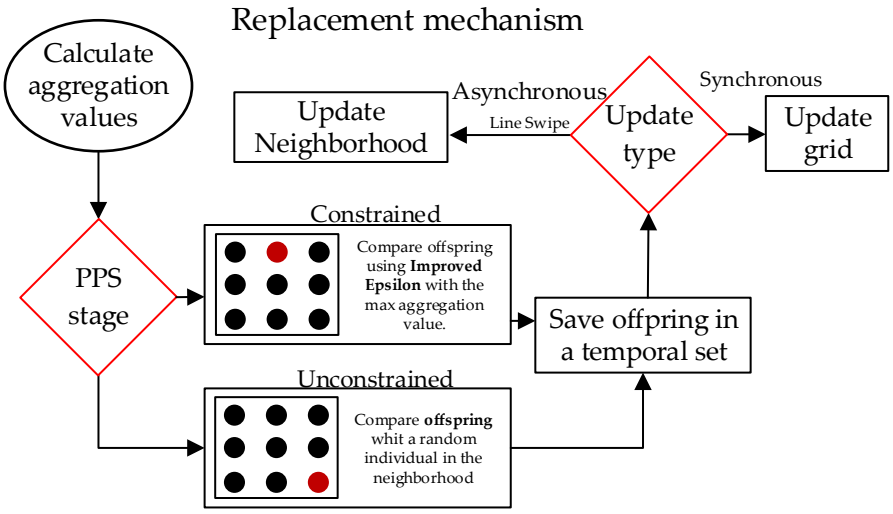


Figure 5.2: Replace mechanism for cMODE/D algorithm.

and 37 of Algorithm 18.

5.1.4 Feedback mechanism

Lines 39 and 40 in Algorithm 18 define an updating mechanism through an external archive. Thus, an external archive is used to store the best solutions in every generation to feedback the population as a good diversity source. This mechanism is inherited from MOCell algorithm [9],

Algorithm 21: $UpdateExternalArchive(archive, P)$

```
1 archive = NDS(archive, 1);
2 if size(Archive) > size(P) then
3     Use crowding distance to rank the archive.
```

Result: *archive*

Algorithm 21 shows how an external archive is filled. Only offspring evolved in every generation are used and they are filtered by using non-dominated sorting to obtain those at the first front. If the archive exceeds population's size, crowding distance is used to truncate it by removing solutions with worst distances. In cMODE/D algorithm, intensity of feedback mechanism is modified by considering a solutions percentage within the population located at a feasible areas within the landscape. If more than 50% of the population is in an infeasible zone, the feedback mechanism is applied through Algorithm 22.

Taking a maximum number of random (δ parameter) solutions from the population and to replace them with random solutions from the external archive, results in an aggressive attempt to introduce good solutions to the population, trying to increase population's feasibility.

Algorithm 22: $Feedback(archive, P)$

```
1 nRep = min(min(δ, size(P)), size(archive));
2 for i=1 to nRep do
3     a, b = rand;
4     P(a) = archive(b);
```

Result: P

When the population exceeds 50% feasibility, the feedback mechanism described in Algorithm 23 is deployed. This feedback criteria, similarly to previous one, randomly select a number of solutions to be replaced. A main difference is that the maximum number

of solutions to replace is $\frac{1}{3}$ of δ parameter, thus $\sigma = \delta \cdot \frac{1}{3}$. This means that when feasibility is high, it is not necessary to have a large number of replacements throughout the population. In this way, a number of good solutions is maintained through final stages of the search without a negative effect. To control which solutions are replaced an extra condition must be fulfilled. It compares a solution in the population against an individual in the external archive. If the solution to be replaced is worse in terms of non-dominance, replacement is made by the external archive solution; otherwise this "replacement" is not considered until the maximum number of replacements is evaluated.

There are two extra cases, one in which all solutions are replaced, and another in which no solution is replaced because current solutions in the population are better ones than those selected from the external archive. Also an intermediate case in where more than one solution and less than σ solutions are replaced.

Algorithm 23: $ReplaceWorstFeedback(archive, P)$

```
1 nRep = min(min(σ, size(P)), size(archive));
2 idxPop = radom(population, nRep);
3 idxArchive = radom(archive, nRep);
4 [rankPop, rankArchive] = NDS(population(idxPop) ⋃ archive(idxArchive));
5 for i=1 to nRep do
6     if rankPop(i) > rankArchive(i) then
7         P(idxPop(i)) = archive(idxArchive(i));
```

Result: P

5.2 Experimental Design

For a thorough empirical assessment, the proposed cMODE/D is compared against two popular algorithmic approaches: MOEA/D and MOCell using the Push and Pull Search as CHT. For experimental deployment, a multiobjective optimization platform called PlatEmo, [74] has been used. It is fully developed in MatLab and provides several different algorithms and benchmark functions as well as a friendly testing environment with widely accepted performance metrics in the area.

Following parameters configured for each algorithmic approach including the proposed cMODE/D are shown in Table 5.1. Parameters for MOEA/D-PPS algorithm are the same to those Authors used in their proposal [28]. MOCellPPS is configured with the same

Table 5.1: Parameters configuration.

<table>
<tr><th>MOEA/D-PPS</th><th>cMODE/D</th><th>MOCELLPPS</th></tr>
<tr><td colspan="3">Population size $N = 300$</td></tr>
<tr><td colspan="3">Population $pm = \frac{1}{n}$, where n is the decision vector dimension
Distribution index is set to 20.</td></tr>
<tr><td colspan="2">DE parameters
$CR = 1.0, F = 0.5$</td><td>GA parameters
Crossover ratio $Pc = 1$,
Distribution index for SBX $Dsc = 20$.</td></tr>
<tr><td>Neighborhood size T=30</td><td colspan="2">Neighborhood size T=9</td></tr>
<tr><td>CHT parameters
$T_c = 800, \alpha = 0.95,$
$\tau = 0.1, cp = 2, l = 20,$
$nr = 2, \rho = 1e-3$</td><td>CHT parameters
$T_c = 800, \alpha = 0.95,$
$\tau = 0.1, l = 20,$
$\rho = 1e-3, nr = 2,$
$\delta = 30, \sigma = 10.$</td><td>CHT parameters
$T_c = 800, l = 20, \rho = 1e-3, \delta = 20$</td></tr>
<tr><td colspan="3">Stopping condition: each algorithm is executed 30 independent runs
and stops when 300,000 function evaluations are reached.</td></tr>
</table>

PPS parameters as MOEA/D-PPS as well as cMODE/D since they share the same CHT. MOCellPPS δ parameter is the same as reported in the original paper [9]. cMODE/D adds two new parameters, δ which is the maximum number of replacements in the feedback mechanism which is set to 10% of the population, and σ as mentioned above is calculated as $\frac{1}{3}$ of δ.

5.2.1 Benchmark functions

For an empirical validation, the LIRCMOP benchmark proposed in [27] was used, which is composed of fourteen restricted functions, as described below:

1.- In problems 5-14, the functions are multiplied by a scalar value, which increases convergence difficulty.

2.- In problems 1-4, one can appreciate those feasible regions are small.

3.- In problems 5 and 6, there are convex and concave Pareto fronts and their actual Pareto fronts are the same as the constrained Pareto fronts.

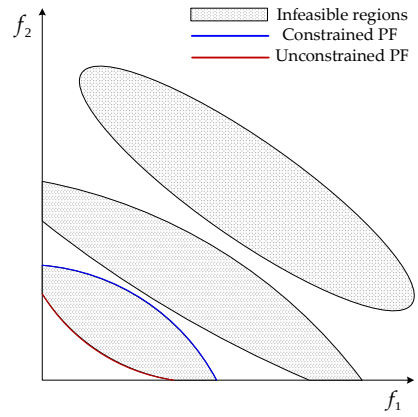
4.- In problems 7 and 8, unconstrained Pareto fronts are located in infeasible regions, and actual Pareto fronts are located at constraints limits.

5.- In problems 9-12, there are two different types of constraints, the first type creates large zones of infeasibility, the second type generates discontinuity between these zones making more difficult the search throughout objective functions landscapes. In addition to solutions located at constraints limits.

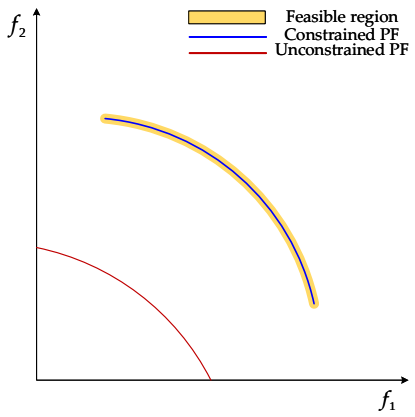
6.- Finally, in problems 13 and 14, there are functions with 3 objectives. Pareto front in function 13 is the same as its unconstrained counterpart. On the other hand, the Pareto front in function 14 is located at the limits of its constraints.

This benchmark proposes a more complex scheme, in the constrained multiobjective optimization area, since it proposes functions with characteristics where finding the optimal front is more demanding, thus representing a more difficult searching context to any algorithm. Crossing infeasible zones in reduced time without losing diversity is a more complex task than in more general benchmarks such as MW [75]. Figure 5.3 shows a graphic representation of each type of problem in the LIRCMOP benchmark.

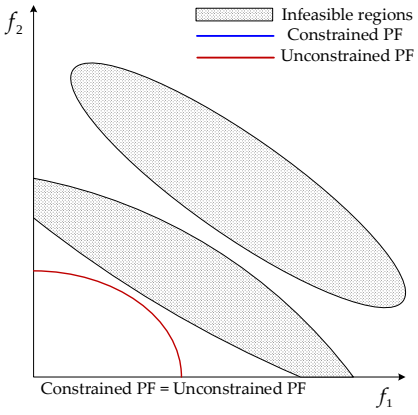
The difference between LIRCMOP and MW is that MW divides its functions into four different types of problems, each one trying to have general characteristics of constrained multiobjective problems, but maintaining different characteristics in itself. In contrast, LIRCMOP is a more complex benchmark specialized in measuring algorithms' performance when dealing with large infeasible regions. The role of differential evolution operator and polynomial mutation is important. Polynomial mutation allows faster convergence when compared to SBX. It achieves good solutions, however this operator is less aggressive in terms of exploration, thus, it takes more generations to locate solutions in feasible areas. Therefore, differential evolution operators had reported good performance in this type of problems [43, 28, 29, 27], Combining differential evolution operator and polynomial mutation on a decentralized cellular population joins together benefits of local exploitation in DE and exploration throughout the cellular population's topology.



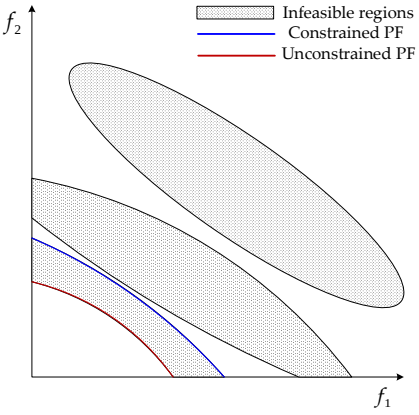
(a) Type 1



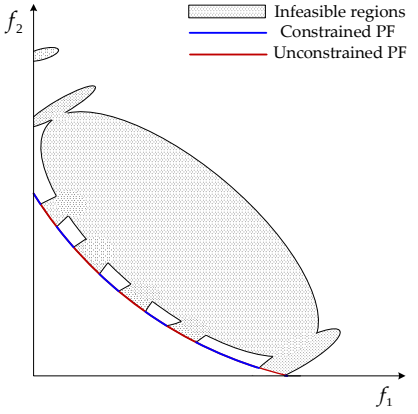
(b) Type 2



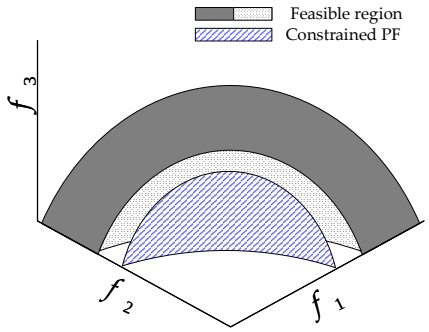
(c) Type 3



(d) Type 4



(e) Type 5



(f) Type 6

Figure 5.3: Constrained multiobjective problems classification of the LIRCMOP benchmark. Each subfigure represents a type of problem regarding the previous list.

5.3 Results Analysis

Tables 5.2 and 5.3 show overall cMODE/D performance results in comparison to previous works in terms of IGD and HV metrics. The proposed approach is empirically compared to two previously proposed approaches known as MOCellPPS and MODEA/D-PPS. cMODE/D is configured with three different types of updating mechanisms: Synchronous (SY), Line Swipe (LS), and Asynchronous (AS). Also, the Wilcoxon test is considered and cMODE/D-AS thus its asynchronous version is taken as a reference.

In Table 5.2, cMODE/D algorithm shows a significantly better performance than MOEA/D-PPS and MOCellPPS in 8 out of 14 problems for IGD metric. On the other hand, for HV metric in Table 5.3 obtained results are shown, where better results in 6 problems are obtained. In these results, Wilcoxon test indicates that one solution is statistically equal.

Using an adaptive feedback mechanism based on population's feasibility impacts the searching process. Applying a random based feedback criterion to this type of problems, where feasible solutions are very important, would stagnate the search in the middle of the evolutionary process. At the beginning of the search, solutions in the population tend to be mostly infeasible solutions, through evolutionary search and as generations elapses, individuals move to feasible landscape's areas. At an early searching stage, a random feedback mechanism works well because it seeks an exploration of the search space. When population feasibility trends towards 50% or higher, exploration does not bring many benefits because it would only create stagnation. Thereby a feedback mechanism to only replace worst solutions aims at maintaining an exploitative search.

Comparing cMODE/D updating mechanism: synchronous, line sweep, and asynchronous. Results shows that the asynchronous version has better performance followed by the line sweep and synchronous version. For a clear perspective in cMODE/D-AS performance, subfigures in Figure 5.4 show different achieved Pareto fronts, these are the average of 30 individual runs.

In Figures 5.4a, 5.4b, 5.4c, and 5.4d, it can be seen that cMODE/-AS and MOEA/D-PPS adjust to the true Pareto fronts (marked in black with + symbol). It is also observed that indeed the CHT perfectly delimits feasible areas. In Figures 5.4e, 5.4f, 5.4g, 5.4h, 5.4j and 5.4k, there is an almost perfect adjustment of cMODE/D-AS and MOEA/D-PPS, although according to IGD and HV metrics, MOEA/D-PPS algorithm achieves superior performance. In those figures, it is observed that difference between both average solutions

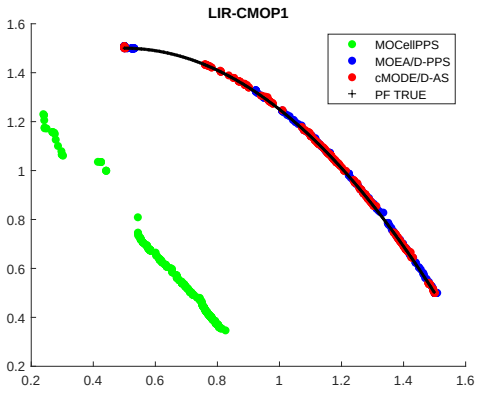
is almost null. On the other hand, in Figures 5.4i and 5.4l, it is observed that cMODE/D-AS cannot reach central zones of the real Pareto front; although difference is minimal. Another algorithmic behaviour observed in results is that MOCellPPS algorithm progress with difficulty when adjusting to benchmark problems, mainly because this algorithm is based only on non-dominance criterion and although follows a cellular structure, this benchmark presents large infeasibility zones. Finally, Figures 5.4m and 5.4n show problems with three objectives. Although, it is difficult to observe clear differences. It can be noticed that MOCellPPS algorithm tends to leave Pareto front zone, while cMODE/D-AS and MOEA/D-PPS adjust correctly.

Table 5.2: IGD metric results for cMODE/D vs MOcell and MOEA/D-PPS on LIRCMOP benchmark functions. cMODE/D updating mechanisms: synchronous (SY), line Swipe (LS) and asynchronous (AS). Columns M and D correspond to the number of objectives and vector's dimension, respectively. Wilcoxon statistical test with cMODE/D-AS as reference is applied, the results marked in grey are the best results obtained from each function, "+/−/≈" means corresponding algorithm is significantly better, worst or not significantly different to cMODE/D-AS.

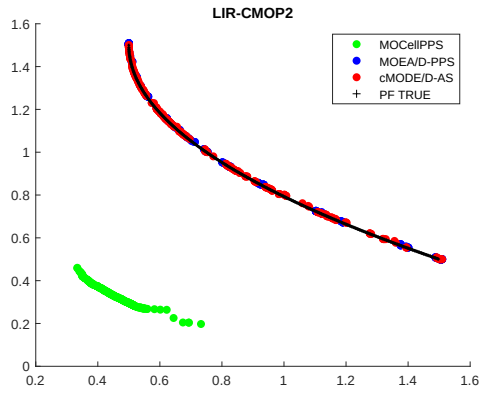
Problem	M	D	MOEA/D-PPS	MOCellPPS	cMODE/D-SY	cMODE/D-LS	cMODE/D-AS
LIRCMOP1	2	30	6.7835e-2 (6.18e-2) −	NaN (NaN)	1.2460e-2 (8.64e-3) +	2.4438e-2 (1.70e-2) ≈	3.1013e-2 (1.78e-2)
LIRCMOP2	2	30	3.8530e-2 (2.28e-2) −	NaN (NaN)	3.6752e-3 (6.15e-3) +	8.6877e-3 (6.39e-3) +	1.2165e-2 (5.82e-3)
LIRCMOP3	2	30	5.6070e-2 (2.61e-2) −	NaN (NaN)	2.1463e-2 (1.99e-2) +	3.6226e-2 (1.95e-2) ≈	4.2382e-2 (2.20e-2)
LIRCMOP4	2	30	5.4034e-2 (3.40e-2) −	NaN (NaN)	1.6025e-2 (2.78e-2) +	2.1427e-2 (1.14e-2) ≈	2.5411e-2 (1.17e-2)
LIRCMOP5	2	30	1.4444e-3 (3.50e-5) +	2.9182e-1 (5.22e-2) −	2.4981e-3 (2.43e-4) −	2.3437e-3 (2.14e-4) ≈	2.2577e-3 (1.47e-4)
LIRCMOP6	2	30	1.3569e-3 (1.81e-5) +	2.8527e-1 (8.26e-2) −	2.3928e-3 (1.01e-4) −	2.2161e-3 (5.57e-5) ≈	2.2100e-3 (6.21e-5)
LIRCMOP7	2	30	3.2460e-3 (2.20e-4) −	8.5224e-1 (2.77e-2) −	4.6115e-3 (1.50e-3) −	7.4497e-3 (1.91e-2) −	2.7559e-3 (6.23e-5)
LIRCMOP8	2	30	3.0345e-3 (1.50e-4) +	9.4820e-1 (6.49e-1) −	4.6087e-3 (1.06e-3) −	3.6130e-3 (1.77e-3) ≈	3.8915e-3 (2.39e-3)
LIRCMOP9	2	30	2.5841e-3 (5.63e-4) +	8.6131e-1 (2.48e-1) −	1.0079e-1 (1.38e-1) ≈	5.3615e-2 (3.45e-2) ≈	5.1393e-2 (3.49e-2)
LIRCMOP10	2	30	2.4831e-3 (7.37e-5) −	2.5493e-1 (1.68e-1) −	1.1058e-2 (2.42e-2) −	5.3257e-3 (1.70e-2) −	2.1117e-3 (1.24e-4)
LIRCMOP11	2	30	3.5157e-2 (5.10e-2) −	8.5336e-1 (3.45e-1) −	5.7200e-2 (7.71e-2) −	4.4555e-2 (8.36e-2) −	1.4085e-2 (3.74e-2)
LIRCMOP12	2	30	4.6194e-3 (1.92e-4) +	9.7970e-1 (5.84e-2) −	1.3455e-1 (3.57e-2) −	1.0438e-1 (6.27e-2) ≈	1.1176e-1 (5.88e-2)
LIRCMOP13	3	30	6.5150e-2 (3.53e-4) −	8.0003e-2 (1.91e-3) −	6.5470e-2 (7.82e-4) −	6.5487e-2 (1.48e-3) ≈	6.4760e-2 (1.11e-3)
LIRCMOP14	3	30	6.7190e-2 (6.37e-4) +	9.8719e-2 (3.71e-3) −	6.8965e-2 (1.71e-3) −	6.8322e-2 (1.11e-3) ≈	6.8020e-2 (1.23e-3)
+/−/≈			6/8/0	0/10/0	4/9/1	1/3/10	

Table 5.3: HV metric results for cMODE/D vs MOcell and MOEA/D-PPS on LIRCMOP benchmark functions. cMODE/D updating mechanisms: synchronous (SY), line Swipe (LS) and asynchronous (AS). Columns M and D correspond to the number of objectives and vector's dimension, respectively. Wilcoxon statistical test with cMODE/D-AS as reference is applied, the results marked in grey are the best results obtained from each function, "+/−/≈" means corresponding algorithm is significantly better, worst or not significantly different to cMODE/D-AS.

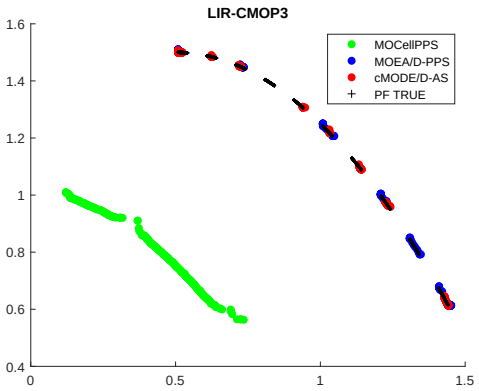
Problem	M	D	MOEA/D-PPS	MOCellPPS	cMODE/D-SY	cMODE/D-LS	cMODE/D-AS
LIRCMOP1	2	30	2.1318e-1 (2.19e-2) −	NaN (NaN)	2.3666e-1 (2.39e-3) +	2.3334e-1 (4.72e-3) ≈	2.3161e-1 (5.34e-3)
LIRCMOP2	2	30	3.4418e-1 (1.13e-2) −	NaN (NaN)	3.6075e-1 (5.81e-3) +	3.5886e-1 (3.04e-3) +	3.5730e-1 (2.60e-3)
LIRCMOP3	2	30	1.8898e-1 (1.05e-2) −	NaN (NaN)	2.0127e-1 (7.54e-3) +	1.9734e-1 (5.77e-3) ≈	1.9598e-1 (5.43e-3)
LIRCMOP4	2	30	2.9180e-1 (1.77e-2) −	NaN (NaN)	3.1073e-1 (1.23e-2) +	3.0827e-1 (4.69e-3) ≈	3.0626e-1 (5.22e-3)
LIRCMOP5	2	30	2.9385e-1 (4.08e-5) +	1.5702e-1 (1.93e-2) −	2.9325e-1 (1.59e-4) −	2.9334e-1 (1.19e-4) ≈	2.9339e-1 (8.68e-5)
LIRCMOP6	2	30	1.9918e-1 (2.63e-5) +	1.1706e-1 (1.78e-2) −	1.9862e-1 (5.34e-5) −	1.9871e-1 (2.94e-5) −	1.9873e-1 (2.89e-5)
LIRCMOP7	2	30	2.9574e-1 (2.46e-4) −	8.4394e-2 (1.81e-2) −	2.9517e-1 (7.61e-4) −	2.9399e-1 (8.46e-3) −	2.9618e-1 (7.82e-5)
LIRCMOP8	2	30	2.9596e-1 (1.08e-4) +	9.2349e-2 (3.96e-2) −	2.9524e-1 (5.52e-4) −	2.9579e-1 (8.45e-4) ≈	2.9568e-1 (1.11e-3)
LIRCMOP9	2	30	5.6735e-1 (6.06e-4) +	2.7505e-1 (5.46e-2) −	5.3720e-1 (3.99e-2) ≈	5.5234e-1 (8.87e-3) ≈	5.5290e-1 (8.88e-3)
LIRCMOP10	2	30	7.0875e-1 (7.81e-5) +	5.7668e-1 (7.32e-2) −	7.0510e-1 (7.90e-3) −	7.0753e-1 (5.29e-3) ≈	7.0858e-1 (1.08e-4)
LIRCMOP11	2	30	6.7553e-1 (3.22e-2) −	4.1270e-1 (9.73e-2) −	6.5876e-1 (4.99e-2) −	6.6677e-1 (5.35e-2) ≈	6.8668e-1 (2.40e-2)
LIRCMOP12	2	30	6.2033e-1 (1.59e-5) +	3.0678e-1 (5.80e-2) −	5.7000e-1 (1.37e-2) −	5.8186e-1 (2.38e-2) ≈	5.7901e-1 (2.24e-2)
LIRCMOP13	3	30	5.6227e-1 (8.16e-4) ≈	5.4807e-1 (1.68e-3) −	5.6070e-1 (1.78e-3) −	5.6080e-1 (2.00e-3) ≈	5.6181e-1 (1.51e-3)
LIRCMOP14	3	30	5.6528e-1 (1.08e-3) +	5.4079e-1 (2.91e-3) −	5.6039e-1 (1.63e-3) −	5.6078e-1 (1.79e-3) ≈	5.6140e-1 (1.29e-3)
+/−/≈			7/6/1	0/10/0	4/9/1	1/2/11	


LIR-CMOP1
MOCellPPS
MOEA/D-PPS
cMODE/D-AS
PF TRUE

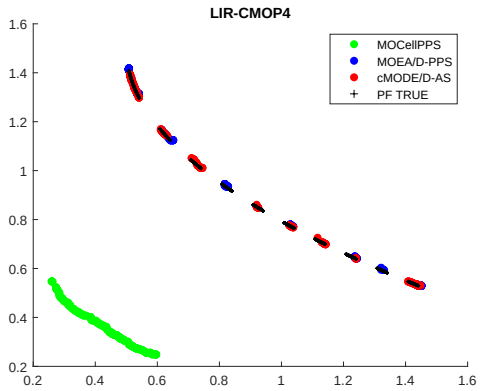
(a)


LIR-CMOP2
MOCellPPS
MOEA/D-PPS
cMODE/D-AS
PF TRUE

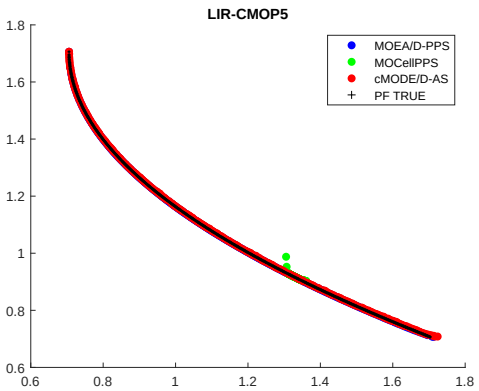
(b)


LIR-CMOP3
MOCellPPS
MOEA/D-PPS
cMODE/D-AS
PF TRUE

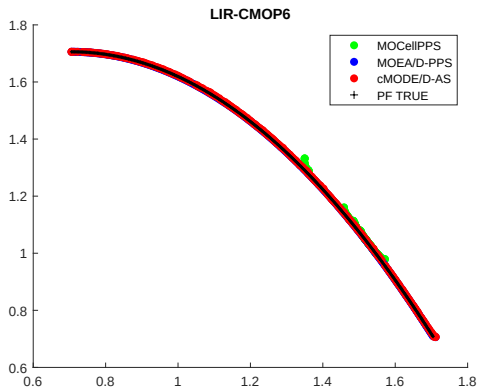
(c)


LIR-CMOP4
MOCellPPS
MOEA/D-PPS
cMODE/D-AS
PF TRUE

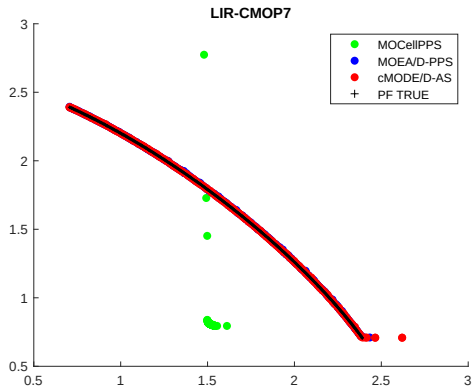
(d)


LIR-CMOP5
MOEA/D-PPS
MOCellPPS
cMODE/D-AS
PF TRUE

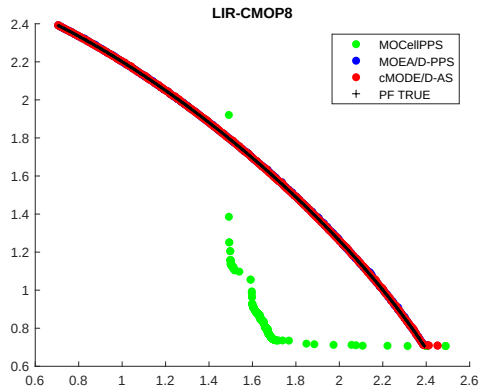
(e)


LIR-CMOP6
MOCellPPS
MOEA/D-PPS
cMODE/D-AS
PF TRUE

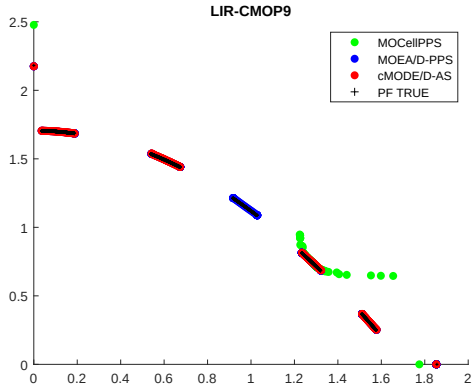
(f)


LIR-CMOP7
MOCellPPS
MOEA/D-PPS
cMODE/D-AS
PF TRUE

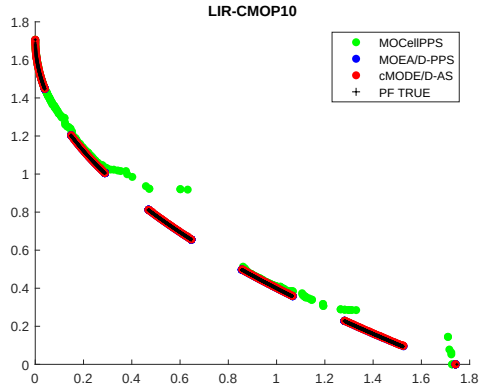
(g)


LIR-CMOP8
MOCellPPS
MOEA/D-PPS
cMODE/D-AS
PF TRUE

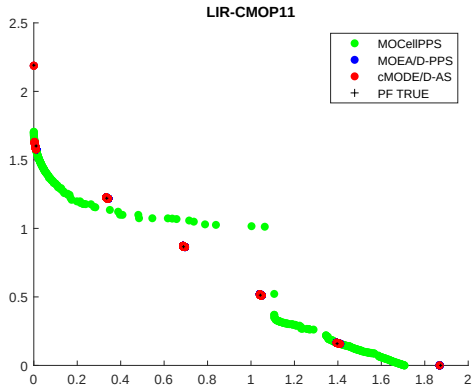
(h)


LIR-CMOP9
MOCellPPS
MOEA/D-PPS
cMODE/D-AS
PF TRUE

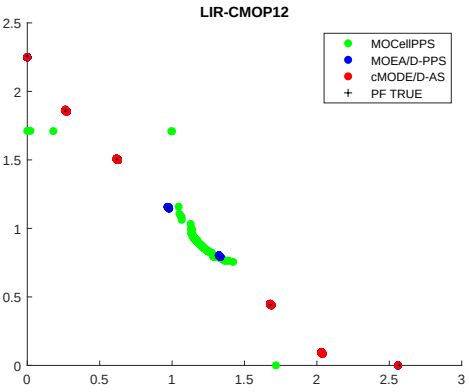
(i)


LIR-CMOP10
MOCellPPS
MOEA/D-PPS
cMODE/D-AS
PF TRUE

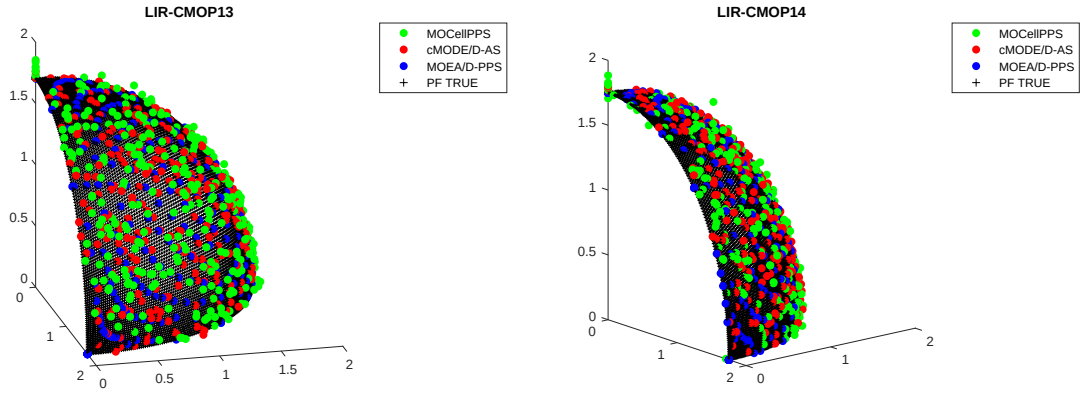
(j)


LIR-CMOP11
MOCellPPS
MOEA/D-PPS
cMODE/D-AS
PF TRUE

(k)


LIR-CMOP12
MOCellPPS
MOEA/D-PPS
cMODE/D-AS
PF TRUE

(l)



(m) (n)

Figure 5.4: Average PFs for MOCell, MOEA/D-PPS and the proposed algorithm cMODE/D-AS on different representative test problems.

5.3.1 Time Analysis

All experimental assessment was carried out on a processor Intel Core I5 2.30 GHz, in 64 bits Windows 10 operating system, with 8GB RAM. The code was written in MATLAB R2019a, all tests were done on PlatEMO [74] platform fully developed in MATLAB [76]. To carry out the time analysis, the total time that the algorithm takes to run was calculated, then the algorithm was executed independently 30 times, in the same way as the IGD and HV metrics; Table 5.4 shows the average in seconds that an algorithm takes to complete the 300,000 assigned evaluations for every problem.

Results show that MOEA/D-PPS presents longest times, this is due to the neighborhoods size used by MOEA/D-PPS which are proportional to the population size. Therefore, as the population size increases, neighborhoods tend to become larger. Moreover, MOEA/D-PPS takes longer to calculate and to perform solution replacement; because initially MOEA/D-PPS takes random indexes to compare between offspring and parents, after this process is performed *nr* number of times until any of both following conditions are satisfied: 1) all solutions are evaluated against offspring, if evolved solutions are worst replacement does not take place; 2) if offspring are better, this process is carried out *nr* number of times until the maximum number of replacements is achieved; this makes the algorithm computationally more expensive.

cMODE/D makes use of *nr* parameter in the same way as MOEA/D-PPS. However, in cMODE/D neighborhoods are not related to the population size. cMODE/D makes use of the Moore neighborhood and although the number of comparisons increases when using *nr* parameter, it does not have the same number of comparisons as in MOEA/D-PPS, since neighbourhood are smaller. Concerning the adaptive feedback mechanism that was proposed, this does not seem to have a great impact on the Algorithm time, since in most of the problems cMODE/D achieves a higher performance than MOEA/D-PPS, this is reflected in Table 5.4. On the other hand, MOCellPPS is based on the cellular EAs model but it does not use decomposition methods thus it is faster than the others. However, MOCellPPS results regarding IGD and HV metrics, its performance is easily overcome by the other two algorithms.

5.3.2 Algorithm complexity for evolutionary algorithms

For an in-depth analysis, a complexity analysis for evolutionary algorithms that have been used on competitions at the Congress on Evolutionary Computation, one of the top con-

Table 5.4: Runtime results for cMODE/D vs MOcell and MOEA/D-PPS on LIRCMOP benchmark functions. cMODE/D updating mechanisms: synchronous (SY), line Swipe (LS) and asynchronous (AS). Columns M and D correspond to the number of objectives and vector's dimension. Wilcoxon statistical test with cMODE/D-AS as reference is applied, the results marked in grey are the best results obtained from each function, "+/−/≈" means corresponding algorithm is significantly better, worst or not significantly different to c/MODE/D-AS.

Problem	M	D	MOEA/D-PPS	MOCellPPS	cMODE/D-SY	cMODE/D-LS	cMODE/D-AS
LIRCMOP1	2	30	3.3902e+2 (2.28e+1) −	1.0627e+2 (7.52e+0) +	1.6021e+2 (1.52e+1) −	1.5193e+2 (9.52e+0) −	1.2308e+2 (5.52e+0)
LIRCMOP2	2	30	3.3146e+2 (2.41e+1) −	1.1315e+2 (1.15e+1) +	1.4886e+2 (7.79e+0) −	1.3952e+2 (8.56e+0) −	1.2256e+2 (6.40e+0)
LIRCMOP3	2	30	4.1209e+2 (9.54e+1) −	1.0794e+2 (1.03e+1) +	1.6889e+2 (1.59e+1) −	1.5708e+2 (9.38e+0) −	1.2642e+2 (6.68e+0)
LIRCMOP4	2	30	3.2951e+2 (2.11e+1) −	1.0395e+2 (1.01e+1) +	1.6170e+2 (1.30e+1) −	1.5943e+2 (1.66e+1) −	1.4420e+2 (1.50e+1)
LIRCMOP5	2	30	2.7316e+2 (1.72e+1) −	9.6394e+1 (6.07e+0) +	1.5441e+2 (9.76e+0) −	1.6359e+2 (1.62e+1) −	1.4270e+2 (1.21e+1)
LIRCMOP6	2	30	2.7628e+2 (1.80e+1) −	9.5473e+1 (6.55e+0) +	1.3945e+2 (5.51e+0) ≈	1.3760e+2 (6.30e+0) ≈	1.4328e+2 (1.62e+1)
LIRCMOP7	2	30	3.0801e+2 (1.89e+1) −	1.1916e+2 (1.49e+1) +	1.6423e+2 (1.18e+1) −	1.5430e+2 (9.93e+0) −	1.3192e+2 (7.56e+0)
LIRCMOP8	2	30	3.4114e+2 (4.58e+1) −	1.0009e+2 (8.32e+0) +	1.4807e+2 (7.12e+0) −	1.3857e+2 (8.36e+0) −	1.1958e+2 (6.74e+0)
LIRCMOP9	2	30	3.3807e+2 (2.35e+1) −	9.7001e+1 (5.31e+0) +	1.4796e+2 (5.97e+0) −	1.3615e+2 (7.18e+0) −	1.2341e+2 (5.27e+0)
LIRCMOP10	2	30	3.3771e+2 (3.11e+1) −	1.2293e+2 (1.38e+1) +	1.3707e+2 (9.30e+0) +	1.3913e+2 (9.08e+0) +	1.4443e+2 (1.48e+1)
LIRCMOP11	2	30	3.3422e+2 (2.23e+1) −	1.2439e+2 (1.55e+1) ≈	1.4501e+2 (7.44e+0) −	1.3415e+2 (7.17e+0) −	1.2057e+2 (6.13e+0)
LIRCMOP12	2	30	3.3140e+2 (1.83e+1) −	1.1161e+2 (9.62e+0) +	1.4881e+2 (7.59e+0) −	1.3808e+2 (7.64e+0) −	1.2370e+2 (6.93e+0)
LIRCMOP13	3	30	2.5252e+2 (1.51e+1) −	1.2614e+2 (1.17e+1) −	1.3416e+2 (7.20e+0) −	1.3237e+2 (7.39e+0) −	1.1802e+2 (6.75e+0)
LIRCMOP14	3	30	2.8202e+2 (2.27e+1) −	1.2130e+2 (9.60e+0) −	1.4493e+2 (8.07e+0) −	1.3373e+2 (6.80e+0) −	1.1346e+2 (5.67e+0)
+/−/≈			0/14/0	11/2/1	1/12/1	1/12/1	

ferences in the area, is carried out [77, 78, 79, 80, 81]. In the evolutionary computing community, the term complexity algorithm was attributed to this procedure although it is not closely related to formal mathematical analysis. The main idea behind this analysis is to define what is the exact time that the algorithm demands without including the evaluation process of the objective functions. Finally, this measure is divided by a value that is defined by the time it takes for a computer to execute a defined number of calculations, to normalize the final result.

The three variables that this analysis involves are defined below.

- A) $T0$ is obtained by evaluating the machine time to perform the following evaluations:
  $for\ i = 1 : 100000$
  $x = 0.55 + i;$
  $x = x + x; x = x/2; x = x * x; x = sqrt(x); x = log(x); x = exp(x); x = x/(x + 2);$
  end

- B) $T1$ is obtained by evaluating the time it takes to evaluate a chosen function n

number of times. In this case, 300,000 evaluations were used and the LIRCMOP3 problem of the LIRCMOP benchmark was chosen.

- (c) $T2$ is the time for the algorithm to solve a function with a maximum number of evaluations, from this value $\hat{T2}$ value is calculated. $\hat{T2}$ is the average running time for 30 executions targeting the LIRCMOP3 benchmark problem.

Once these three values are calculated, the following calculation is carried out $(\hat{T2} - T1)/T0$. The value obtained from this operation shows the total time it takes for the algorithm to complete all evolutionary stages (selection, recombination, mutation, and replacement) while excluding the time it takes to evaluate the objective function, which is normally the bottleneck calculation in most optimization evolutionary algorithms. In Table 5.5, achieve values are presented.

Table 5.5: Results of the operation $(\hat{T2} - T1)/T0$.

Algorithm	$(\hat{T2} - T1)/T0$
MOEA/D-PPS	4.634859581169369e+03
MOCellPPS	1.084289493744679e+03
cMODE/D-SY	1.795804360844909e+03
cMODE/D-LS	1.657937411206620e+03
cMODE/D-AS	1.300020334499597e+03

From Table 5.5, it can be seen that these results effectively confirm results shown in Table 5.4, with MOEA/D-PPS being the slowest-running algorithm. In the case of MOCellPPS algorithm, as explained in Section 5.3.1, since it is based on non-dominance and is simpler than cMODE/D, it is faster, but its performance is overcome in IGD and HV metrics.

It can be noticed that depending on update criterion that cMODE/D applies, time complete all evaluations increases, being the asynchronous version the fastest of the three, following by Line Sweep version and finally the Synchronous one. This behavior is given by how the grid is updated, synchronous version must keep in a temporary variable all solutions to replace, and wait to finish the generation to be able to update. On the contrary, asynchronous criterion does not use this temporary variable, while as a middle case, Line Sweep criterion uses the temporary variable but the current generation does not need to evolve completely to be able to update.

5.4 Summary

In this chapter cMODE/D algorithm was presented. It uses advantages of cellular evolutionary algorithms as well as differential evolution in combination with MOEA/D decomposition principles. Similar to cMOGA/D, cMODE/D is focused on solving constrained MOPs, therefore Push to Pull search and Improved Epsilon CHT were implemented. cMODE/D also uses different update schemes: synchronous, line sweep, and asynchronous. Its feedback mechanism adapts to population's feasibility, thus having two ways to be applied. If population's feasibility is lower to 50%, a random feedback mechanism of δ replacements is applied. If feasibility increases by more than 50%, the number of replacements is decreased and replacement is only applied if solutions in the population are dominated by those in the external file.

From tests carried out on the LIRCMOP benchmark, it can be seen that cMODE/D in its asynchronous version obtains superior performance than MOEA/D-PPS and MOCellPPS algorithms in terms of IGD and HV metrics. A time analysis was also conducted, concluding that cMODE/D is at least twice as fast as MOEA/D-PPS, and shows a competitive time against MOCellPPS.

The next chapter summarizes the research of this thesis and presents its conclusions. Furthermore, some possible guidelines for future work are presented.

Chapter 6

Conclusions and Future work

In this research two novel algorithmic approaches to tackle constrained multiobjective problems are presented. A cellular multiobjective Genetic Algorithm based on decomposition (cMOGA/D) and a cellular multiobjective Differential Evolution based on decomposition (cMODE/D) are designed and empirically assessed. Both take advantage of structural properties in cellular evolutionary algorithms and combines core concepts of well established MOEA/D. A thorough experimental assessment showed that cMOGA/D outperforms previously proposed approaches when tackling standard benchmark problems such as those in Ma and Wang (MW) benchmark test [75]. On the other hand, cMODE/D, combines the same algorithmic structure but deploys those evolutionary operations corresponding to differential evolution. Empirical evaluation presented in this thesis research shows promising results, in a more specialized and more complex benchmark Large infeasible regions constrained multiobjective problems (LIRCMOP) [27].

Decentralized population schemes implemented in cMOGA/D and cMODE/D were promising optimization tools not only to tackle constrained MOPs but also to improve the performance of centralized approaches. Therefore, at this point, it can be concluded that the hypothesis initially proposed in this thesis was confirmed. Besides, all the objectives set and the expected contributions were successfully reached.

An important aspect in cMOGA/D and cMODE/D is the possibility of implementing different updating types mechanisms that structural properties allow to define which can also positively impact the searching process. Moreover, using aggregation values, which are a main concept in MOEA/D, to assess solutions updating allows to achieve a significantly improved performance.

cMOGA/D and cMODE/D makes use of decentralized and structured populations promoting local solutions exploitation and global landscapes exploration. Moreover, in order to preserve solutions diversity, three different updating mechanisms have been thor-

oughly evaluated in this thesis research, together with an external feedback criterion brought from MOCellPPS, a previous closely related algorithmic approach. In addition, in cMODE/D, the feedback mechanism was extended and linked to the population's feasibility population's, in order to soften solutions' feedback that could affect the final stages of the evolutionary process.

Testing cMOGA/D on MW benchmark allowed to verify its algorithmic ability to solve CMOPs with a number of characteristics such as low feasible rations, convergence difficulty and high dimensional decision vectors. Thus, cMOGA/D was developed focusing on solving general constrained multiobjective problems. On the other hand, cMODE/D was developed to deal with more complex problems such those in LIRCMOP benchmark. In LIRCMOP, large infeasible regions in objective spaces are created, this is the key characteristic of this benchmark.

SBX operator in cMOGA/D, creates close located solutions, because SBX generates offspring from only two parents. However, polynomial mutation adds a small exploration factor, insufficient to tackle this kind of benchmarks. Therefore, even though solutions are closely created, they can be discarded by constraint handling techniques, or in other cases, they can have a slow convergence, and not reaching feasible areas. This impact during the searching process is emphasized by using a decentralized population such as cellular ones; because solutions are slowly spread throughout the mesh more slowly. Differential evolution operator, when directly operating decision vectors, involves a more aggressive search and adds polynomial mutation to somehow strengthen exploration phase. DE is fast at convergence in addition to using a decentralized population via a cellular topology, which allows a balance between exploitation and exploration searching phases. These characteristics make DE operations suitable for this type of benchmarks where exploration takes a very important role in the early stages of the search. Therefore, cMODE/D integrates differential evolution and polynomial mutation, as well as adaptive feedback, which was developed to obtain an improvement in this type of benchmarks.

A significant impact of having a cellular structure in cMOGA/D and cMODE/D algorithms is using different updating criteria. Centralized or panmictic based multiobjective evolutionary algorithms commonly use a synchronous updating mechanism which generate all evolved solutions at once and then decide by truncation or selection. In algorithms with parallel populations, specifically in algorithms with cellular structures and thus decentralized populations, updating criterion is key since neighborhoods are overlapped and each solution itself is the center of a local neighborhood. Thus solutions spread-

ing throughout the toroidal mesh depends on the established updating mechanism. Synchronous updating presents a slower solutions spreading because it does not allow new solutions (offspring) recombination with current solutions (parents). In contrast, asynchronous updating is a special case because immediately after solutions within a neighborhood undergo evolution (selection, crossover, mutation, and replacement), new solution(s) from that neighbourhood are available for mating with those currently available in close by neighbourhoods.

Empirical achieved results show the impact of the updating criteria in terms of processing times to execute a number of evaluations. Synchronous updating takes longer when compared to asynchronous criterion which is fastest while line sweep shows an intermediate behaviour.

For 85% of the problems tested in MW, cMOGA/D outperformed previously proposed approaches MOEA/D-PPS and MOCellPPS. On the other hand in 57% of the problems in the LIRCMOP benchmark cMODE/D outperforms previous widely validate approaches. In the same way, cMODE/D algorithm with asynchronous update criterion has a slightly higher performance when targeting LIRCMOP benchmark, which is highly complex. Also according to obtained results regarding processint times, cMOGA/D and cMODE/D proposals obtained very promising results, performing the same number of evaluations in less than half the time as the MOEA/D-PPS algorithm.

6.1 Future work

Future work aims to extend the algorithm to target real-world problems, as well as into many-objectives benchmarks to find out other mechanisms based on structural characteristics of cellular evolutionary algorithms to improve their searching. Also attacking problems with high dimensionality, which is currently a trending topic in the evolutionary computation community.

Bibliography

[1] D. M. Pedroso, M. R. Bonyadi, and M. Gallagher, "Parallel evolutionary algorithm for single and multi-objective optimisation: Differential evolution and constraints handling," *Applied Soft Computing*, vol. 61, pp. 995–1012, dec 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494617305458

[2] E. Alba and B. Dorronsoro, *Cellular Genetic Algorithms*, ser. Operations Research/Computer Science Interfaces Series. Boston, MA: Springer US, 2008, vol. 42. [Online]. Available: http://link.springer.com/10.1007/978-0-387-77610-1

[3] S. Mirjalili, "Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems," *Neural Computing and Applications*, vol. 27, no. 4, pp. 1053–1073, may 2016.

[4] K. Deb, "Multi-objective Optimization," in *Search Methodologies*. Boston, MA: Springer US, 2014, pp. 403–449. [Online]. Available: http://link.springer.com/10.1007/978-1-4614-6940-7{_}15

[5] H. Fukumoto and A. Oyama, "Benchmarking multiobjective evolutionary algorithms and constraint handling techniques on a real-world car structure design optimization benchmark problem." Association for Computing Machinery (ACM), jul 2018, pp. 177–178.

[6] S. Garcia and C. T. Trinh, "Comparison of Multi-Objective Evolutionary Algorithms to Solve the Modular Cell Design Problem for Novel Biocatalysis," *Processes*, vol. 7, no. 6, p. 361, jun 2019. [Online]. Available: https://www.mdpi.com/2227-9717/7/6/361

[7] C. Ramirez-Atencia and D. Camacho, "Constrained multi-objective optimization for multi-UAV planning," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 6, pp. 2467–2484, jun 2019.

[8] F. Luna and E. Alba, *Parallel Multiobjective Evolutionary Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 1017–1031. [Online]. Available: https://doi.org/10.1007/978-3-662-43505-2_50

[9] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba, "MOCell: A cellular genetic algorithm for multiobjective optimization," *International Journal of Intelligent Systems*, vol. 24, no. 7, pp. 726–746, jul 2009.

[10] J. Zheng, C. Lu, and L. Gao, "Multi-objective cellular particle swarm optimization for wellbore trajectory design," *Applied Soft Computing Journal*, 2019.

[11] Y. Sato, M. Sato, H. Goto, and M. Miyakawa, "Distributed NSGA-II Sharing Extreme Non-Dominated Solutions for Constrained Knapsack Problems," in *2018 International Conference on Control, Artificial Intelligence, Robotics & Optimization (ICCAIRO)*. IEEE, may 2018, pp. 197–202. [Online]. Available: https://ieeexplore.ieee.org/document/8698400/

[12] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*, ser. Natural Computing Series. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. [Online]. Available: http://link.springer.com/10.1007/978-3-662-44874-8

[13] A. Brabazon, M. O'Neill, and S. McGarraghy, *Natural Computing Algorithms*, ser. Natural Computing Series. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. [Online]. Available: www.springer.com/series/http://link.springer.com/10.1007/978-3-662-43631-8

[14] D. B. Fogel, T. Back, and Z. Michalewicz, *Evolutionary computation. Vol. 1, Basic algorithms and operators*. Institute of Physics Pub, 2000.

[15] A. E. Eiben, C. H. M. Kemenade, and J. N. Kok, "Orgy in the computer: Multi-parent reproduction in genetic algorithms," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer Verlag, 1995, vol. 929, pp. 934–945. [Online]. Available: http://link.springer.com/10.1007/3-540-59496-5{_}354

[16] S. Sastry, D. E. Goldberg, and G. Kendall, *Search Methodologies*, E. K. Burke and G. Kendall, Eds. Boston, MA: Springer US, 2014. [Online]. Avail-

able: https://link.springer.com/content/pdf/10.1007{%}2F978-1-4614-6940-7.pdfhttp://link.springer.com/10.1007/978-1-4614-6940-7

[17] K. Deb and R. B. Agrawal, "Simulated Binary Crossover for Continuous Search Space," *Complex Systems*, vol. 9, pp. 1–34, 1994. [Online]. Available: citeulike-article-id:2815748http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.26.8485

[18] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, dec 2007.

[19] Hui Li and Qingfu Zhang, "Multiobjective Optimization Problems With Complicated Pareto Sets, MOEA/D and NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 284–302, apr 2009. [Online]. Available: http://ieeexplore.ieee.org/document/4633340/

[20] C. Garcia-Garcia, M.-G. Martínez-Peñaloza, and A. Morales-Reyes, "cMOGA/D: a novel cellular GA based on decomposition to tackle constrained multiobjective problems." in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. New York, NY, USA: ACM, jul 2020, pp. 1721–1729. [Online]. Available: https://doi.org/10.1145/3377929.3398137

[21] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," Tech. Rep. 2, 2002.

[22] W. Zheng, Y. Tan, L. Meng, and H. Zhang, "An improved MOEA/D design for many-objective optimization problems," *Applied Intelligence*, vol. 48, no. 10, pp. 3839–3861, oct 2018. [Online]. Available: https://link.springer.com/content/pdf/10.1007{%}2Fs10489-018-1183-5.pdfhttp://link.springer.com/10.1007/s10489-018-1183-5

[23] F. Luna, G. R. Zavala, A. J. Nebro, J. J. Durillo, and C. A. Coello, "Distributed multiobjective metaheuristics for real-world structural optimization problems," *Computer Journal*, vol. 59, no. 6, 2014.

[24] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, 2001.

[25] K. Deb, K. Sindhya, and T. Okabe, "Self-adaptive simulated binary crossover for real-parameter optimization," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation - GECCO '07*. New York, New York, USA: ACM Press, 2007, p. 1187. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1276958.1277190

[26] K. A. De Jong, *Evolutionary Computation : A Unified Approach*. Cambridge, MA: The MIT Press, 2006.

[27] Z. Fan, W. Li, X. Cai, H. Huang, Y. Fang, Y. You, J. Mo, C. Wei, and E. Goodman, "An improved epsilon constraint-handling method in MOEA/D for CMOPs with large infeasible regions," *Soft Computing*, vol. 23, no. 23, pp. 12 491–12 510, 2019. [Online]. Available: https://doi.org/10.1007/s00500-019-03794-x

[28] Z. Fan, W. Li, X. Cai, H. Li, C. Wei, Q. Zhang, K. Deb, and E. Goodman, "Push and pull search for solving constrained multi-objective optimization problems," *Swarm and Evolutionary Computation*, vol. 44, pp. 665–679, feb 2019.

[29] Z. Fan, Y. Fang, W. Li, X. Cai, C. Wei, and E. Goodman, "MOEA/D with angle-based constrained dominance principle for constrained multi-objective optimization problems," *Applied Soft Computing Journal*, 2019.

[30] K. Deb and ayan Deb, "Analysing mutation schemes for real-parameter genetic algorithms," *International Journal of Artificial Intelligence and Soft Computing*, vol. 4, no. 1, p. 1, 2014.

[31] R. Storn and K. Price, "Differential Evolution-A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces," Tech. Rep., 1997.

[32] J. C. Bansal and N. R. Pal, *Swarm and Evolutionary Computation*. Cham: Springer International Publishing, 2019, pp. 1–9. [Online]. Available: https://doi.org/10.1007/978-3-319-91341-4_1

[33] C. A. Coello Coello, G. B. Lamont, and D. A. Van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*, ser. Genetic and Evolutionary Computation Series. Boston, MA: Springer US, 2007. [Online]. Available: http://link.springer.com/10.1007/978-0-387-36797-2

[34] K. Deb, *Evolutionary and Swarm Intelligence Algorithms*, ser. Studies in Computational Intelligence, J. C. Bansal, P. K. Singh, and N. R. Pal, Eds. Cham: Springer International Publishing, 2019, vol. 779. [Online]. Available: http://www.springer.com/series/7092http://link.springer.com/10.1007/978-3-319-91341-4{_}6http://link.springer.com/10.1007/978-3-319-91341-4

[35] M. T. M. Emmerich and A. H. Deutz, "A tutorial on multiobjective optimization: fundamentals and evolutionary methods," *Natural Computing*, vol. 17, no. 3, pp. 585–609, sep 2018. [Online]. Available: http://link.springer.com/10.1007/s11047-018-9685-y

[36] C. M. Fonseca and P. J. Fleming, "Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization," 1993.

[37] J. Knowles and D. Corne, "The pareto archived evolution strategy: a new baseline algorithm for pareto multiobjective optimisation," in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, vol. 1, 1999, pp. 98–105 Vol. 1.

[38] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach," *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, vol. 3, no. 4, pp. 257–271, 1999.

[39] E. Zitzler, M. Laumanns, and L. Thiele, "Spea2: Improving the strength pareto evolutionary algorithm," Tech. Rep., 2001.

[40] N. Srinivas and K. Deb, "Multiobjective optimization using nondominated sorting in genetic algorithms," *Evolutionary Computation*, vol. 2, pp. 221–248, 1994.

[41] S. Kukkonen and J. Lampinen, "GDE3: The third Evolution Step of Generalized Differential Evolution." Institute of Electrical and Electronics Engineers (IEEE), dec 2005, pp. 443–450.

[42] M. A. Jan and R. A. Khanum, "A study of two penalty-parameterless constraint handling techniques in the framework of MOEA/D," *Applied Soft Computing Journal*, 2013.

[43] Z. Fan, F. Yi, W. Li, J. Lu, X. Cai, and C. Wei, "A comparative study of constrained multi-objective evolutionary algorithms on constrained multi-objective optimization

problems," in *2017 IEEE Congress on Evolutionary Computation, CEC 2017 - Proceedings*. Institute of Electrical and Electronics Engineers Inc., jul 2017, pp. 209–216.

[44] H. R. Cheshmehgaz, H. Haron, and A. Sharifi, "The review of multiple evolutionary searches and multi-objective evolutionary algorithms," *Artificial Intelligence Review*, vol. 43, no. 3, pp. 311–343, 2013.

[45] G. Luque and E. Alba, *Parallel Genetic Algorithms*, ser. Studies in Computational Intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, vol. 367. [Online]. Available: http://link.springer.com/10.1007/978-3-642-22084-5

[46] E. Cantú-Paz, *Efficient and Accurate Parallel Genetic Algorithms*, ser. Genetic Algorithms and Evolutionary Computation. Boston, MA: Springer US, 2001, vol. 1. [Online]. Available: http://link.springer.com/10.1007/978-1-4615-4369-5

[47] E. Alba and M. Tomassini, "Parallelism and evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 443–462, oct 2002. [Online]. Available: http://ieeexplore.ieee.org/document/1041554/

[48] M. Tomassini, *Spatially Structured Evolutionary Algorithms*, ser. Natural Computing Series. Berlin/Heidelberg: Springer-Verlag, 2005. [Online]. Available: http://link.springer.com/10.1007/3-540-29938-6

[49] F. Ferrucci, P. Salza, and F. Sarro, "Using Hadoop MapReduce for Parallel Genetic Algorithms: A Comparison of the Global, Grid and Island Models," *Evolutionary Computation*, vol. 26, no. 4, pp. 535–567, dec 2018. [Online]. Available: https://www.mitpressjournals.org/doi/abs/10.1162/evco{_}a{_}00213

[50] F. Luna, A. J. Nebro, and E. Alba, "Parallel Evolutionary Multiobjective Optimization," in *Parallel Evolutionary Computations*. Springer Berlin Heidelberg, 2006, vol. 22, pp. 33–56. [Online]. Available: http://link.springer.com/10.1007/3-540-32839-4{_}2

[51] E. Alba, B. Dorronsoro, F. Luna, and P. Bouvry, "A cellular multi-objective genetic algorithm for optimal broadcasting strategy in metropolitan manets," in *19th IEEE International Parallel and Distributed Processing Symposium*, 2005, pp. 8 pp.–.

[52] D. Sudholt, *Parallel Evolutionary Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 929–959. [Online]. Available: https://doi.org/10.1007/978-3-662-43505-2_46

[53] D. Simoncini, S. Verel, P. Collard, and M. Clergue, "Anisotropic selection in cellular genetic algorithms," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation - GECCO '06*. New York, New York, USA: ACM Press, 2006, p. 559. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1143997.1144098

[54] ——, "Centric selection a Way to Tune the Exploration/Exploitation Trade-off David," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation - GECCO '09*. New York, New York, USA: ACM Press, 2009, p. 891. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1569901.1570023

[55] B. Xu, W. Duan, H. Zhang, and Z. Li, "Differential evolution with infeasible-guiding mutation operators for constrained multi-objective optimization," *Applied Intelligence*, jul 2020. [Online]. Available: http://link.springer.com/10.1007/s10489-020-01733-0

[56] B. Y. Qu and P. N. Suganthan, "Constrained multi-objective optimization algorithm with an ensemble of constraint handling methods," *Engineering Optimization*, vol. 43, no. 4, pp. 403–416, 2011.

[57] F. Qian, B. Xu, R. Qi, and H. Tianfield, "Self-adaptive differential evolution algorithm with -constrained-domination principle for constrained multi-objective optimization," *Soft Computing*, vol. 16, no. 8, p. 1353–1372, 2012.

[58] H. Jain and K. Deb, "An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part ii: handling constraints and extending to an adaptive approach," *IEEE Transactions on evolutionary computation*, vol. 18, no. 4, pp. 602–622, 2013.

[59] K. Li, K. Deb, Q. Zhang, and S. Kwong, "An evolutionary many-objective optimization algorithm based on dominance and decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 5, pp. 694–716, 2014.

[60] Y. Tian, R. Cheng, X. Zhang, F. Cheng, and Y. Jin, "An indicator-based multiobjective evolutionary algorithm with reference point adaptation for better versatility," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 4, pp. 609–622, 2017.

[61] Z.-Z. Liu, Y. Wang, and B.-C. Wang, "Indicator-based constrained multiobjective evolutionary algorithms," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2019.

[62] Z. Yang, X. Cai, and Z. Fan, "Epsilon constrained method for constrained multiobjective optimization problems: some preliminary results," in *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, 2014, pp. 1181–1186.

[63] M. Asafuddoula, T. Ray, R. Sarker, and K. Alam, "An adaptive constraint handling approach embedded moea/d," in *2012 IEEE Congress on Evolutionary Computation*, 2012, pp. 1–8.

[64] Y. G. Woldesenbet, G. G. Yen, and B. G. Tessema, "Constraint handling in multiobjective evolutionary optimization," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 3, pp. 514–525, 2009.

[65] Y. Yang, J. Liu, S. Tan, and H. Wang, "A multi-objective differential evolutionary algorithm for constrained multi-objective optimization problems with low feasible ratio," *Applied Soft Computing Journal*, 2019.

[66] R. Cheng, Y. Jin, M. Olhofer, and B. Sendhoff, "A reference vector guided evolutionary algorithm for many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 5, pp. 773–791, 2016.

[67] K. Li, R. Chen, G. Fu, and X. Yao, "Two-archive evolutionary algorithm for constrained multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 2, pp. 303–315, 2019.

[68] W. Ning, B. Guo, Y. Yan, X. Wu, J. Wu, and D. Zhao, "Constrained multi-objective optimization using constrained non-dominated sorting combined with an improved hybrid multi-objective evolutionary algorithm," *Engineering Optimization*, vol. 49, no. 10, pp. 1645–1664, oct 2017.

[69] H.-L. Liu, F. Gu, and Q. Zhang, "Decomposition of a multiobjective optimization problem into a number of simple multiobjective subproblems," *IEEE transactions on evolutionary computation*, vol. 18, no. 3, pp. 450–455, 2013.

[70] A. Arias Montaño, C. A. Coello Coello, and E. Mezura-Montes, "pMODE-LD+SS: An Effective and Efficient Parallel Differential Evolution Algorithm for Multi-Objective Optimization," in *Parallel Problem Solving from Nature, PPSN XI*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, no. 1, pp. 21–30. [Online]. Available: http://link.springer.com/10.1007/978-3-642-15871-1{_}3

[71] J. J. Durillo, A. J. Nebro, F. Luna, and E. Alba, "Solving Three-Objective Optimization Problems Using a New Hybrid Cellular Genetic Algorithm," 2008, pp. 661–670. [Online]. Available: http://link.springer.com/10.1007/978-3-540-87700-4{_}66

[72] J. D. Knowles, L. Thiele, and E. Zitzler, "A tutorial on the performance assessment of stochastic multiobjective optimizers," *TIK-Report*, vol. 214, 2006.

[73] A. Jaimes and C. Coello Coello, "MRMOGA: Parallel Evolutionary Multiobjective Optimization using Multiple Resolutions," in *2005 IEEE Congress on Evolutionary Computation*, vol. 3. IEEE, dec 2005, pp. 2294–2301. [Online]. Available: http://ieeexplore.ieee.org/document/1554980/

[74] Y. Tian, R. Cheng, X. Zhang, and Y. Jin, "PlatEMO: A MATLAB platform for evolutionary multi-objective optimization," *IEEE Computational Intelligence Magazine*, vol. 12, no. 4, pp. 73–87, 2017.

[75] Z. Ma and Y. Wang, "Evolutionary Constrained Multiobjective Optimization: Test Suite Construction and Performance Comparisons," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 6, pp. 972–986, dec 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8632683/

[76] MATLAB, *Version 9.6.0.1335978 (R2019a) Update 8*. Natick, Massachusetts: The MathWorks Inc., 2019. [Online]. Available: https://la.mathworks.com/products/matlab.html

[77] J. Liang, B.-Y. Qu, and P. N. Suganthan, "Problem Definitions and Evaluation Criteria for the CEC 2014 Special Session and Competition on Single Objective Real-Parameter Numerical Optimization," Tech. Rep., 2014.

[78] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization," *KanGAL report*, vol. 2005005, no. 2005, p. 2005, 2005.

[79] J. Liang, T. P. Runarsson, E. Mezura-Montes, M. Clerc, P. N. Suganthan, C. C. Coello, and K. Deb, "Problem definitions and evaluation criteria for the cec 2006 special session on constrained real-parameter optimization," *Journal of Applied Mechanics*, vol. 41, no. 8, pp. 8–31, 2006.

[80] J. Liang, B. Qu, P. Suganthan, and A. G. Hernández-Díaz, "Problem definitions and evaluation criteria for the cec 2013 special session on real-parameter optimization," *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University, Singapore, Technical Report*, vol. 201212, no. 34, pp. 281–295, 2013.

[81] J. Liang, B. Qu, P. Suganthan, and Q. Chen, "Problem definitions and evaluation criteria for the cec 2015 competition on learning-based real-parameter single objective optimization," *Technical Report201411A, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore*, vol. 29, pp. 625–640, 2014.

Acronyms

- **EA** Evolutionary Algorithm.
- **MOP** Multiobjective Optimization Problem.
- **CMOP** Constrained Multiobjective Optimization Problem.
- **CHT** Constraint Handling Technique.
- **MOEA** Multiobjective Evolutionary Algorithm.
- **GECCO** The Genetic and Evolutionary Computation Conference.
- **EC** Evolutionary Computation.
- **AI** Artificial Intelligence.
- **GA** Genetic Algorithm.
- **PMX** Partially Mapped Crossover.
- **SBX** Simulated Binary Crossover.
- **DE** Differential Evolution.
- **MOEA/D** Multiobjective Evolutionary Algorithm based on Decomposition.
- **POS** Pareto Optimal Set.
- **NSGA** Non-dominated Sorting Genetic Algorithm.
- **NSGA-II** Non-dominated Sorting Genetic Algorithm II.

- **NDS** Non-dominated Sorting.
- **NSGA-III** Non-dominated Sorting Genetic Algorithm III.
- **MOGA** Multiobjective Genetic Algorithm.
- **PAES** The Pareto Archived Evolution Strategy.
- **SPEA** A Strength Pareto Evolutionary Algorithm
- **SPEA-2** A Strength Pareto Evolutionary Algorithm II
- **MOCell** MultiObjective Cellular Genetic Algorithm.
- **GDE3** The third Evolution Step of Generalized Differential Evolution.
- **MOEA/D-DE** Multiobjective Evolutionary Algorithm based on Decomposition with Differential Evolution.
- **CDP** Constraint Dominance Principle.
- **SR** Stochastic Ranking.
- **PPS** Push and Pull Search.
- **SIMD** Single-Instruction Multiple-Data.
- **MIMD** Multiple-Instruction, Multiple-Data.
- **MPI** Message Passing Interface.
- **RMI** Java-Remote Method Invocation.
- **CORBA** Common Object Request Broker Architecture.
- **GPU** Graphics Processing Unit.
- **cMOGA** Cellular Multi-Objective Genetic Algorithm.
- **cEA** Cellular Evolutionary Algorithms.
- **GD** Generational Distance.
- **IGD** Inverted Generational Distance.

- **HV** Hypervolume.
- **IMDE** Infeasible-guiding Mutation Differential Evolution.
- **TNK** abbreviation of Tanaka.
- **OSY** Osyczka initials.
- **CTP** Constrained Test Problems.
- **NCTP** New Constrained Test Problems.
- **MODE-ECHT** Multi-objective Differential Evolution - Ensemble of Constraint Handling Methods.
- **SADE-CD** Self-Adaptive Differential Evolution Algorithm with - Constrained-Domination Principle.
- **MOEA/DD** Many-Objective Optimization Algorithm based on Dominance and Decomposition.
- **ARMOEA** Adaptive Reference Point Multiobjective Evolutionary Algorithim.
- **MOEA/D-IEpsilon** Multiobjective Evolutionary Algorithm based on Decomposition with Improved Epsilon.
- **ICMOEA** Indicator-based Constrained Multiobjective Evolutionary algorithm.
- **MOEA/D-Epsilon** Multiobjective Evolutionary Algorithm based on Decomposition with Epsilon Constrained.
- **MOEAD/D-SR** Multiobjective Evolutionary Algorithm based on Decomposition with Stochastic Ranking.
- **MOEA/D-CDP** Multiobjective Evolutionary Algorithm based on Decomposition with Constraint Dominance Principle.
- **C-MOEA/D** Adaptive Constraint Handling Multiobjective Evolutionary Algorithm based on Decomposition.

- **NSGA-II-CDP** Non-dominated Sorting Genetic Algorithm II with Constraint Dominance Principle.
- **LIR-CMOP** Large Infeasible Regions- Constrained Multiobjective Problems.
- **ACDP** Angle Constraint Dominance Principle.
- **MOEA/D-ACDP** Multiobjective Evolutionary Algorithm based on Decomposition- Angle Constraint Dominance Principle.
- **SP** SPacing-based Muliobjective evolutionary Algorithm.
- **RVEA** Reference Vector guided Evolutionary Algorithm.
- **CTAEA** Constrained Two-archive Evolutionary Algorithm.
- **MODE-SaE** Multi-objective Differential Evolution- Self-adaptively Epsilon.
- **CNS** Constrained Non-dominated Sorting.
- **cMOEA/H** Constrained Hybrid Multiobjective Optimization Algorithm.
- **pMODE-LS+SS** Parallel Multi-Objective Differential Evolution Algorithm Incorporating Local Dominance and Scalar Selection Mechanisms.
- **ZDT** Zitzler, Deb and Thiele.
- **DTLZ** Deb, Thiele, Laumanns, and Zitzler.
- **CellDE** Cellular Differential Evolution.
- **MRMOGA** Multiple Resolutions Multiobjective Genetic Algorithm.
- **MOCPSO** Multiobjective cellular Particle Swarm Optimization Algorithm.
- **MOPSO** Multiobjective Particle Swarm Optimization Algorithm.
- **MW** Ma and Wang.

- **MOCellPPS** MultiObjective Cellular Genetic Algorithm with Push and Pull Search.
- **cMOGA/D** Cellular Multiobjective Genetic Algorithm based on Decomposition.
- **cMOGA/D-SY** Cellular Multiobjective Genetic Algorithm based on Decomposition -Synchronous.
- **cMOGA/D-AS** Cellular Multiobjective Genetic Algorithm based on Decomposition -Asynchronous.
- **cMOGA/D-LS** Cellular Multiobjective Genetic Algorithm based on Decomposition -Line Swipe.
- **MOEA/D-PPS** Multiobjective Evolutionary Algorithm based on Decomposition-Push and Pull Search.
- **PlatEMO** Platform for Evolutionary Multi-Objective Optimization.
- **MATLAB** Matrix Laboratory.
- **cMODE/D** Cellular Multiobjective Differential Evolution Algorithm based on Decomposition.
- **cMODE/D-SY** Cellular Multiobjective Differential Evolution Algorithm based on Decomposition-Synchronous.
- **cMODE/D-AS** Cellular Multiobjective Differential Evolution Algorithm based on Decomposition-Asynchronous.
- **cMODE/D-LS** Cellular Multiobjective Differential Evolution Algorithm based on Decomposition-Line Swipe.