Chapter 3

Metaheuristics

Nowadays, researchers employ three different approaches to solve optimization problems: exact algorithms, heuristics, and metaheuristics. The main advantage of using exact algorithms is that they offer the guarantee of finding global optimum for a given problem. However, they are not directly applicable to the most of real problems (NP-hard) because the required execution time grows exponentially with the size of the problem. In contrast, the non structured heuristics are usually pretty fast, but generally they offer non optimal solutions. Furthermore, the adaptation of this approach to the most of problems is complex. Finally, the metaheuristics are generic methods which offer good solutions, even global optimum, within a reasonable computing time.

This chapter is aimed at establishing the foundations needed to understand the metaheuristics algorithms used to address optimization problems. First, in Section 3.1, we offer some important definitions to understand the topic of optimization problems. Then, Section 3.2 briefly introduces a representative set of optimization techniques. Finally, Section 3.3 presents in more detail the used algorithms to solve the problem addressed in this project.

3.1 Definition of Metaheuristic

Optimization problems consist of the search for a best configuration of a set of variables to achieve some goals [35]. Generally, optimization problems can be seen as the search for the best solution, or at least good enough, to a given problem. Currently they are using optimization techniques in fields as diverse as networking, economics or logistics. Humans are constantly solving optimization problems, such as what

39
route to take to get from one place to another, how to organize our schedule, etc. As these problems have small dimension, we can process them easily by using our brain. However, when the optimization problems become larger and more complex it appears the necessity of using specialized tools and the computational power of computers.

This section is intended to define different concepts that will be used throughout the chapter and whole memory. Beginning with the formal definition of optimization. Assuming, without loss of generality, in the case of minimization, we can define an optimization problem as:

**Definition 1 (Optimization Problem):** The pair $P = (S, f)$ is an optimization problem where $S \neq \emptyset$ represents the search space (or solution space), and $f$ is an objective function (or fitness), defined as:

$$f : S \rightarrow \mathbb{R}$$

The search space $S$ is given by a set of variables $X = \{x_1, x_2, ..., x_n\}$ and their domains that are respectively $D_1, D_2, ..., D_n$. Thus, solving an optimization problem consist in finding a solution $x^*_i \in S$ such that

$$f(x^*_i) \leq f(x_i), \forall x_i \in S$$

Above, the optimization problem is presented as a minimization problem, since according to the next equation [52] there exist a direct relation between maximization and minimization problems:

$$\max\{f(x_i) \mid x_i \in S\} \equiv \min\{-f(x_i) \mid x_i \in S\}$$

Optimization problems can be classified depending on the domain where $S$ belongs. According to this classification, we can define binary ($S \subseteq \mathbb{B}^*$), integer ($S \subseteq \mathbb{N}^*$), real ($S \subseteq \mathbb{R}^*$), and mixed or heterogeneous ($S \subseteq (\mathbb{B} \cup \mathbb{N} \cup \mathbb{R})^*$) optimization problems.

The importance of the existing optimization problems has prompted the development of multiple methods to solve them along the history of Information Technology. A brief classification of these techniques is shown in Figure 3.1. The more general classification which divides these techniques into exact and approximate. The **exact**
3.1. DEFINITION OF METAHEURISTIC

metaheuristics, which are based on the mathematical extraction of the optimal solution, or an exhaustive search until the optimum is found, guarantee the optimality of the solution obtained. These techniques present some drawbacks, however. The time they require, though bounded, is generally very large, especially for \textit{NP-complex} problems. Furthermore, it is not always possible to find such an exact technique for every problem. This makes exact techniques not to be the right choice in many occasions, since both their time and memory requirements can become unreasonably high for large problems. Therefore, \textbf{approximate methods} have been often used by the research community in the last few decades. These methods sacrifice the guarantee of finding the optimum in favor of providing some satisfactory solution within reasonable time.

![Optimization algorithms classification](image)

**Figure 3.1:** Optimization techniques classification.

Among approximate algorithms, one can find two types: \textbf{ad hoc heuristics} and \textbf{metaheuristics}. We focus this chapter on the latter. Ad hoc heuristics can in turn be divided between \textbf{constructive heuristics} and \textbf{local search methods}.

Constructive heuristics are usually the swiftest methods. They construct a solution from scratch by iteratively incorporating components until a complete solution is obtained, which is returned as the algorithm output. Finding some constructive heuristic can be easy in many cases, but the obtained solutions are of low quality. In fact, designing one such method that actually produces high quality solutions is a nontrivial task, since it mainly depends on the problem, and requires thorough understanding of it. For example, in problems with many constraints it could happen that many partial solutions do not lead to any feasible solution.
CHAPTER 3. METAHEURISTICS

The neighborhood of a given solution \( x_i \), denoted as \( N(x_i) \), is the set of solutions (neighbors) that can be reached from \( x_i \) through the use of a specific modification operator, generally referred to as movement, (see Definition 2). A local optimum is a solution equal or better than any other solution in its own neighborhood (see Definition 3). The process of exploring the neighborhood, finding and keeping the best neighbor, is repeated in a process until the local optimum is found. Complete exploration of a neighborhood is often unapproachable, therefore some modification of the generic scheme has to be adopted. Depending on the movement operator, the neighborhood varies and so does the manner of exploring the search space, simplifying or complicating the search process as a result [86].

Definition 2 (Neighborhood): Let \((S, f)\) an optimization problem. A neighborhood structure is a function:

\[
N : S \longrightarrow S
\]  

(3.4)

that assigns to every \( x_i \in S \) a set of neighbors \( N(x_i) \in S \). \( N(x_i) \) is called the neighborhood of \( x_i \) [15].

The introduction of a neighborhood structure enables us to define the concept of locally optimal solutions.

Definition 3 (Local optimum): Let \((S, f)\) an optimization problem, and \( S_{i'} \in S \) the neighborhood of \( x_{i'} \in S \) \((N(x_{i'}) = S_{i'})\), \( x_{i'} \) is a local optimum (minimum) if it satisfies the following inequality:

\[
f(x_{i'}) \leq f(x_i), \forall x_i \in S_i
\]  

(3.5)

Finally, three decades ago came a new stream research within the approximate algorithms. It basically tries to combine basic heuristic methods in higher level frameworks aimed at efficiently and effectively exploring a search space. These methods are called metaheuristics, but they were often called modern heuristics. To this type of algorithms belong the tabu search (TS), genetic algorithms (GA), differential evolution (DE), and simulated annealing (SA), among others. Since its emergence the term metaheuristic has been defined in different ways. Osman and Laporte proposed a widely used definition in 1996 [106], which is presented following:
A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions.

The fundamental properties which characterize the set of metaheuristic algorithms are the following ones [47]:

- Metaheuristics are higher level strategies that guide the search process.
- The goal is to efficiently explore the search space in order to find (quasi-)optimal solutions.
- Metaheuristic algorithms are approximate and generally non-deterministic.
- The basic concepts of metaheuristics permit an abstract level of description.
- Metaheuristics are not problem-specific.
- Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.
- Todays more advanced metaheuristics use search experience (embodied in some form of memory) to guide the search.

### 3.2 Metaheuristics Classification

There are different ways to define a metaheuristics taxonomy. Depending on the characteristics selected to differentiate among them, several classifications are possible, each of them being the result of a specific point of view. Next, we briefly summarize the most important ways of classifying metaheuristics [15]:

- **Nature inspired (bio-inspired) vs. non-nature inspired**: Generally, it is the most natural way to classify metaheuristics, since it is based on the origins of the algorithm. It takes into account whether their models have been inspired by Nature or not. There are bio-inspired algorithms, like Genetic Algorithms (GA) and Ant Colony Algorithms (ACO), and non nature-inspired ones such as Tabu Search (TS) and Iterated Local Search (ILS). This classification is not very meaningful since the emergence of hybrid algorithms.
• **Population-based vs. single point search (trajectory):** In this case, the characteristic used for the classification is the number of solutions used at the same time. On the one hand, single point search algorithms work on a single solution describing a trajectory in the search space during the search process. They encompass local search-based metaheuristics, like Variable Neighborhood Search (VNS), Tabu Search (TS), and Iterated Local Search (ILS). On the other hand, population-based methods work on a set of solutions (points) called population. Figure 3.1 shows graphically some examples of this taxonomy.

• **Static vs. dynamic objective function:** The algorithms which keep the objective function given in the problem during the whole process are called metaheuristics with static objective function. However, there are other algorithms with dynamic objective function, like Guided Local Search (GLS), which modify the fitness function during the search, incorporating information collected during the search process to escape from local optimum.

Other characteristics can be used for classification of different metaheuristics, like the number of neighborhood structures and the use or not of memory. In the following subsections, we present briefly some of the most representative metaheuristics grouping them into trajectory or population based algorithms.

### 3.2.1 Trajectory-based Metaheuristics

The main characteristic of trajectory-based metaheuristics is that they start from an initial solution (point) which is updated exploring the neighborhood, describing a trajectory. These metaheuristics emerge to improve local search methods extending them adding mechanisms to escape from local minimum. Such methods need new stop conditions different from finding local minimum, they use more complex and elaborated ones. The most commonly used are that the algorithm stop when a certain number of iterations is reached, when the found solution has an acceptable quality, or when it is detect a deadlock in the search process. Next, we describe briefly some metaheuristics based on trajectory.

• **The Simulated Annealing (SA) is probably the first algorithm that applies an explicit strategy to escape from local minimum.** It was described in 1983 [73], its origins lie in a statistical mechanism called metropolis [89]. The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their
initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one. To avoid local minimum, the algorithm allows to choose a solution with worse fitness than the current solution. Over successive iterations of the algorithm is chosen, from the current solution $s$, a solution $s'$ in the neighborhood $N(s)$. If $s'$ has better quality than $s$, then $s$ is replaced by $s'$ as the current solution. However, if $s'$ is worse, then it is accepted with a certain probability that depends on the current temperature $T$ and the difference between the two fitness solutions $(f(s) - f(s'))$, in the case of minimization.

- **Tabu Search (TS)** is known as one of metaheuristics that have been applied with greater success in solving classical and real optimization problems. The fundamentals of this method were introduced in [49], and both, technique and components, were specified in [50]. It uses flexible structures of memory, which permit exploit the search information more thoroughly than by rigid memory or memory-less systems, conditions for strategically constraining and freeing the search process (emboided in tabu restrictions and aspiration criteria), and memory functions of varying time spans for intensifying and diversifying the search (reinforcing attributes historically found good and driving the search into new regions).

- **Variable Neighborhood Search (VNS)** [92] is a relatively recent metaheuristic which relies on iteratively exploring neighbourhoods of growing size to identify better local optima. VNS has been applied to a wide variety of problems both from combinatorial and continuous optimization. VNS method escapes from the current local minimum $s'$ by by initiating other local searches from starting points sampled from a neighbourhood of $s'$ ($s'' \leftarrow N(s')$) which increases its size iteratively until a local minimum better than the current one is found. These steps are repeated until a given termination condition is met.

More precisely, the first step of this method is to define a set of neighborhoods for which we can use different methods. Then, successive iterations consist of three phases: the choice of candidate, a phase of improvement, and finally, movement. In the first step, it randomly chooses a neighbor $s'$ of $s$ using the $k-th$ neighborhood. This solution ($s'$) is used as the starting point for local search of the second phase, obtaining $s''$. When the process of improvement is finished, the new solution $s''$ is compared with the original $s$. If it is better, $s''$ becomes the current solution and the neighborhoods counter is initialized.
(k ← 1). Otherwise, the process is repeated by using following neighborhood
(k ← k + 1). Local search is intensifying over the method and changing
neighborhood it can be seen as the way of diversification.

• **Greedy Randomized Adaptive Search Procedure (GRASP)** [121] is a simple
metaheuristic that combines constructive heuristics with local search. GRASP
is an iterative procedure with two phases: first, a solution is constructed, sec-
ond, the solution undergoes an improvement process. The improved solution
is the final result of the search process. A randomized heuristic is used for the
construction of the solution in the first phase. Step by step, different com-
ponents c are added to the partial solution s^p, initially empty. Each added
component is randomly selected from a restricted list of candidates (RCL).
This list is a subset of N(s^p), the set of permitted components for the partial
solution s^p. The components of the solution in N(s^p) are sorted according to
some problem dependent function η in order to generate the list. The RCL
list consists of the α best components in the set. In the extreme case of α = 1,
only the best component found is added to the list, thus resulting in a greedy
construction method. In the other extreme, α = ||N(s^p)||, the component is
chosen in a totally random way among all available components. Hence, α is
a key parameter that determines how the search space is going to be sampled.
The second phase of the algorithm consists in a local search method to im-
prove the previously generated solutions. A simple local search method can
be employed, or some more complex technique like SA or TS.

• **Iterated Local Search (ILS)** [130] is a metaheuristic which involves the repeated
application of a local search algorithm applied to the candidate solutions found
by a broader search process that involves a biased random walk through the
search space. ILS is based on a simple but effective idea. The algorithm
works by first selecting a starting point for the search either randomly or
via a domain specific construction heuristic. Then, the starting position is
optimized using a local search strategy. The algorithm loop involves three
steps: a perturbation of the current solution, the application of the local search
to the perturbation, and an acceptance decision of whether or not the locally
optimizing candidate solution should replace the current working solution for
the search. The acceptance of new solutions depending on the search history
and characteristics of the new local minimum.
3.2. METAHEURISTICS CLASSIFICATION

3.2.2 Population-based Metaheuristics

In contrast with the previous methods, population-based algorithms are characterized by working with a set of solutions (called population) in each iteration. Here we introduce some representative population-based metaheuristics:

- **Evolutionary Algorithms (EA)** are based on the postulates of the evolutionary theory, that is, they are inspired by biological evolution processes: selection, recombination, mutation, and reproduction [11]. This family of techniques presents an iterative and stochastic process that operates on a population of individuals (solutions).

EAs start by generating the population randomly or by means of some heuristic seeding procedure. The global structure of the process consists of three main phases (emulating the biological evolution): selection, production, and replacement, that are repeated until meeting certain stop criteria. In the first phase individuals of the population are chosen to be recombined during the production phase. Usually, the individuals are chosen according to their fitness. To increase the diversity, a mutation operator is applied to the individuals generated previously. Finally, a new population is created by using the current one and/or the best individuals generated, giving way to the next generation of the algorithm. Different algorithms have been proposed based on this general scheme. These proposals can be classified into three main categories that were developed independently, they are: the *Evolutionary Programming (EP)* proposed by Fogel [43], the *Genetic Algorithms (GA)* introduced by Holland [59], and the *Evolution Strategies (ES)* submitted by Rechenberg [120].

- **Estimation of Distribution Algorithms (EDAs)** [94] have similar behaviors to the previously presented EAs, and many authors even consider EDAs as a special kind of EA. Like EAs, EDAs operate on a population of candidate solutions, but, unlike them, do not use recombination and mutation to generate the new solutions, but a probability distribution mechanism instead.

Graphic probabilistic models are commonly used tools to represent in an efficient manner the probability distributions when working with EDAs. Some authors [95, 20, 126] propose using bayesian networks to represent the probability distributions in discrete domains, while gaussian networks are most often employed for continuous domains [139].
• **Scatter Search (SS)** [51] is a metaheuristic whose principles were presented in [48]. The main contribution of this algorithm is the idea of maintaining a relatively small set of tentative solutions (called the reference set or RefSet). This set of solutions is characterized by its quality and variety (distant in the search space). This set is divided into subsets of solutions to which we apply an operation of recombination and other improvements.

Scatter Search consists of five main component processes:


2. An *Improvement Method* to transform a trial solution into one or more enhanced trial solutions.

3. A *Reference Set Update Method* to build and maintain a reference set consisting of the *b best* solutions found organized to provide efficient accessing by other parts of the solution procedure.

4. A *Subset Generation Method* to operate on the reference set, to produce a subset of its solutions as a basis for creating combined solutions.

5. A *Solution Combination Method* to transform a given subset of solutions produced by the Subset Generation Method into one or more combined solutions.

Finally, note that it is currently receiving much attention from the research community [79].

• **Ant Colony Optimization (ACO)** [39] algorithm is a probabilistic technique for solving computational problems which can be reduced to finding good paths through graphs. It is based on the behavior of ants seeking a path between their colony and a source of food. The original idea has since diversified to solve a wider class of numerical problems, and as a result, several problems have emerged, drawing on various aspects of the behavior of ants. The behavior of the ants is as follows: Initially, ants explore randomly the area near to the nest. As soon as an ant finds food, it comes back to the nest. In turn, this ant deposits a chemical called *pheromone*, creating pheromone trails. If other ants find such a path, they are likely not to keep travelling at random, but to instead follow the trail, returning and reinforcing it if they eventually find food. The indirect communication between ants using pheromone trails enables them to find the shortest path between the nest and the food. ACO tries to solve
3.3. **ALGORITHMS USED IN THIS WORK**

optimization problems by simulating this behavior. The technique is based on two main steps: *Construct Ant Solutions* and *Update Pheromones* (pheromone matrix).

- **Particle Swarm Optimization (PSO)** [70] is a population based metaheuristic inspired in the social behavior of organisms such as bird flocking and fish schooling. It is based on the factors that influence the decisions of an agent that is part of a set of similar agents. In PSO, each candidate solution to the problem is called *particle* and the population of particles is called *swarm*. During the iterative process, each particle adjusts its position (state) according to its own experience, and according to the experience of a neighboring particle, making use of the best position encountered by itself and its neighbor.

- **Differential Evolution (DE)** [129] is a non-deterministic method based on a population of individuals who are real vectors that represent the solutions in the search space. It was developed to optimize real (float) parameters of a real valued function. The generation of new individuals is carried out by applying differential operators of mutation and crossover to the individuals that are selected randomly. By differential mutation the proportional difference of the randomly chosen parents is added to a third individual, also chosen randomly. After the mutation, a recombination operator is applied over each individual (target) to generate an intermediate individual (trial). Finally, taken into account the fitness values, a selection operator decides the acceptance (or not) of trial individuals for the new generation.

### 3.3 Algorithms used in this Work

In this section, we describe the five metaheuristic algorithms used in this study. Specifically, they are *Simulated Annealing (SA), Genetic Algorithm (GA), Evolutionary Strategy (ES), Particle Swarm Optimization (PSO),* and *Differential Evolution (DE).*

We select these five algorithms because they are able to work on continuous (real valued) search spaces. Also, these techniques were selected with the aim of experimenting with different population structures, as well as different reproduction mechanisms.
3.3.1 Simulated Annealing (SA)

Simulated Annealing was one of the first metaheuristic algorithms with an explicit strategy to escape from local minimum. The basic idea is to allow movements to select solutions worse than the current solution. The probability of performing such a movement decreases during the search process (see Algorithm 1). The operation of a typical SA is summarized in Figure 3.2.

The whole process starts by generating an initial solution $s$ (randomly or using some heuristic) and starting the temperature parameter ($T$). The algorithm works iteratively keeping a single tentative solution $s$ at any time. In every iteration, a new solution $s' \in N(s)$ is generated from the previous one (Line 5), and either replaces it or not depending on an acceptance criterion (Lines 6-10). The acceptance criterion works depending on the fitness values ($f(s)$ and $f(s')$) and temperature $T$. $s'$ replaces $s$ as current solution if $f(s) < f(s')$ ($s'$ has better quality). However, in the case of $f(s) \geq f(s')$, $s'$ replaces it with probability $prob$ (Equation 3.6). This probability depends on the difference between their quality ($f(s') - f(s)$) values and $T$ (Line 9). This acceptance criterion provides the way of escaping from local optima.

$$prob(T, s, s') = \frac{2}{1 + e^{\frac{f(s') - f(s)}{T}}}$$  (3.6)
3.3. ALGORITHMS USED IN THIS WORK

**Algorithm 1** Simulated Annealing (SA)

1: \( s \leftarrow \text{generateSolution}() \)
2: \( \text{initialize}(T) \)
3: \( \text{evaluate}(s) \)
4: \( \textbf{while} \ g < \text{maxGenerations} \text{ or stopCondition}() \ \textbf{do} \)
5: \( s' \leftarrow \text{neighborSolution}(N(s)) \)
6: \( \textbf{if} \ (f(s') < f(s)) \ \textbf{then} \)
7: \( s \leftarrow s' \quad // \text{ } s' \text{ replaces } s \text{ as current solution} \)
8: \( \textbf{else} \)
9: \( \quad \text{accept } s' \text{ as current solution with a probability } prob(T, s, s') \)
10: \( \textbf{end if} \)
11: \( \text{update}(T) \)
12: \( \textbf{end while} \)

The temperature value \( T \) is updated (generally decreasing) during the search. Thus, at the beginning the probability of accepting new solutions are high and it gradually decreases along the iterations, following a cooling schedule (Line 11). The algorithm ends by selecting only those solutions that improve the current one \( s \). This process is based on annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local optimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one.

The algorithm is a result of the combination of two different processes: random step and iterative improvement. In the first phase, exploration of the search space becomes more random and erratic, producing movements to worst solutions. However, as it iterates, the erratic component gradually decreases the search converging to a optimum (local). The selection of solutions that do not improve the current one is controlled by two factors: the difference between the fitness functions and temperature (see Equation 3.6). First, if we fix the temperature, the higher the difference \( f(s') - f(s) \), less probability to move from \( s \) to \( s' \). Moreover, the higher the temperature \( T \), the more probability to accept \( s' \) as a new solution.

The selection of a suitable cooling strategy is crucial for the behavior of the algorithm. The cooling strategy \( Q \) defines the temperature \( T \) for each moment \( k \), \( T_{k+1} = Q(T_k, k) \). The most used one is \( T_{k+1} = \alpha \times T_k \), where \( \alpha \in (0, 1) \). The cooling strategy and the initial temperature must be adapted to problems because the cost
of escaping a local optimum depends on the structure of the search space. There are many ways leading to different variants of SA as: Fast SA (FSA), Very Fast SA (VFSA) or Adaptive SA (ASA) [17]. Ingber [64] proposed an empirical solution which starts with an initial random sampling of search space and $T_0$ is calculated from the average and the standard deviation of the values of the objective function.

SA has been successfully applied to many optimization problems: the Quadratic Assignment Problem (QAP) [36] and the textit{Job Shop Scheduling} (JSS) [134]. In the field of wireless mobile networks has been used in problems of optimizing MANET routing protocols [84] or the clustering of the nodes [82].

### 3.3.2 Genetic Algorithm (GA)

Genetic Algorithms are possibly the most widespread subclass of EAs. They were conceived by John Holland [60]. As EA, it applies the natural selection principles to solve optimization problems. During successives generations, the solutions evolve to the optimum according to these principles. The evolution of these solutions depends largely on an adequate codification of them.

GAs work with a population of individuals, each of which represents a feasible solution to a given problem. Each individual is assigned a value associated with the goodness of this solution (fitness). Similarly, in Nature, the fitness could be seen as a value of of the adaptation and the effectiveness to compete for certain resources for biological organisms. The higher individual adaptation to the problem, the grater the probability to be selected to breed, i.e., crossing its genetic material with another individual. This crossing produces new individuals descendants of the above ones, which share some characteristics of their parents. So, spread the genetic material of the best individuals in successive generations (see Figure 3.4). If the algorithm is designed properly, the population will converge to an optimum solution to the problem.

The algorithm manipulates a collection $p$ of individuals (the population), each of which comprises one or more chromosomes. These chromosomes allow each individual represent a potential solution for the problem under consideration. An encoding/decoding process is responsible for performing this mapping between chromosomes and solutions. Chromosomes are divided into smaller units termed genes. The different values a certain gene can take are called the alleles for that gene.
3.3. ALGORITHMS USED IN THIS WORK

The genes are encoded in a string of symbols (numbers or letters). There are various representations (see Figure 3.3) such as the use of binary numbers in BCD or Gray codes, the use of real or integer, etc. Most of the time, a correct coding is the key to a good resolution of the problem. Typically, codification is static, but in cases of numerical optimization, the number of bits dedicated to encode a parameter can vary.

![Figure 3.3: Examples of gens encoding.](image)

The algorithm (see Algorithm 2) is an iterative process which starts by generating an initial population of solutions. This is typically addressed by randomly generating the desired number of solutions. When the alphabet used for representing solutions has low cardinality, this random initialization provides a more or less uniform sample of the solution space. Then, applying the genetic operators (selection (Line 4), recombination (Line 5), mutation (Line 6), and replacement (Line 8)) on this population.

**Algorithm 2 Genetic Algorithms (GA)**

1: $p^0 \leftarrow \text{initializePopulation}()$
2: $\text{evaluation}(p^0)$
3: while $g < \text{maxGenerations}$ or $\text{stopCondition()}$ do
4: $p_g^s \leftarrow \text{selection}(p^g)$
5: $p_g^r \leftarrow \text{recombination}(p_g^s)$
6: $p_g^m \leftarrow \text{mutation}(p_g^r)$
7: $\text{evaluation}(p_g^m)$
8: $p^{g+1} \leftarrow \text{replacement}(p_g^m)$
9: end while
Selection is responsible for the competition aspects of individuals in the population. In fact, replacement can be intuitively regarded as the complementary application of the selection operation. There exist different techniques to perform the selection, the most popular ones are:

- **Roulette-wheel Selection**: In these methods, the probability of selecting an individual for breeding ($p_i$) is proportional to its fitness ($f_i$) as Equation 3.7 shows. This selection technique allows the selection of the best individuals with greater probability, but at the same time the worst ones could be selected, maintaining the diversity of the population.

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \quad (3.7)$$

A drawback of this procedure is the appearance of an individual whose fitness is much better than the remaining individuals. Such *super-individuals* can quickly take over the population. To avoid this, the best option is using a *non-fitness-proportionate* mechanism.

- **Ranking Selection**: In this technique, the individuals are ranked according to their fitness (best first, worst last), and later selected, e.g., by using *Stochastic Universal Sampling* [12] which follows the probability showed in Equation 3.8.

$$p_i = \frac{1}{|P|} \left[ \eta^- + (\eta^+ - \eta^-) \frac{i - 1}{|P| - 1} \right] \quad (3.8)$$

Where $p_i$ is the probability of selecting the $i$th best individual, and $\eta^- + \eta^+ = 2$. 

Figure 3.4: Operation of a typical GA.
3.3. ALGORITHMS USED IN THIS WORK

- *Tournament Selection*: In this case, $\alpha$ individuals are sampled at random, and the best of them is selected for reproduction. This is repeated as many times as needed. The tournament size ($\alpha$) determines the selective pressure, the higher $\alpha$, the stronger the selective pressure. These unproportionate selection methods have the advantage of being insensitive to fitness scaling problems and to the sense of optimization (maximization or minimization). The main drawback is the need to set the parameter $\alpha$, although the widely used value is two (*binary tournament*).

After selecting the parents, their chromosomes are combined, using recombination and mutation operators.

Recombination is a process that models information exchange among several individuals (typically two). This is done by constructing new solutions using the information contained in a number of selected parents. The resulting individuals are called *offspring*. There exists several recombination operators, the most used for *bitstrings* are the classical *single-point crossover* and *two-point crossover* (see Figure 3.5). Usually, the crossover operator does not apply to all pairs of individuals who have been selected, they are applied to a certain proportion of them with a certain probability. In the case where the operator does not apply crossover, the offspring is obtained by simply duplicating the parents.

![a) Single-point Crossover](image1)

![b) Two-point Crossover](image2)

Figure 3.5: Widely used crossover operators.

The mutation operator is applied to each individual of the offspring separately. This operator injects new material in the population, but at low rate, otherwise the search would degrade to a random walk in the solution space. As it was the case for recombination, the choice of a mutation operator depends on the representation.
used, and it is used just in a portion of the chromosome. In bitstrings (and in general, in linear strings) mutation is done by randomly substituting the symbol contained at a certain position by a different symbol. Figure 3.6 shows the mutation of the fifth gene of the chromosome.

![Mutation operator over one gene.](image)

Figure 3.6: Mutation operator over one gene.

After the evaluation of the offspring (Line 7), the replacement step is carried out to keep the population size constant. To do so, some individuals from the population have to be substituted by some of the individuals created during reproduction (offspring). This can be done in several ways [102]:

- **Replacement-of-the-worst**: the population is sorted according to fitness, and the new individuals replace the worst ones from the population.

- **Random replacement**: the individuals to be replaced are selected at random.

- **Tournament replacement**: a subset of $\alpha$ individuals is selected at random, and the worst one is selected for replacement. Notice that if $\alpha = 1$ we have random replacement.

- **Direct replacement**: the offspring replace their parents.

There are two ways to do this task which define two different genetic algorithms: **steady state GA (ssGA)** and **generational GA (genGA)**. In the first one, every individual asynchronously replaced after generation. genGA defines the replacement of generations, that is, all individuals are updated at once at the end of each generation.

GAs have been thoroughly used in many domains obtaining satisfactory results. For example, GAs have been applied to solve problems such as the scheduling and timetabling [116], the software project management [7], the ARN structures prediction [140], etc. GAs have been applied to the field of MANET/VANET networks to solve problems such as the location of base stations [85], the energy consumption of routing protocols [142] or the optimization of broadcasting protocols [6].

56
3.3.3 Evolutionary Strategies (ES)

Evolutionary Strategies (ES) are Evolutionary Algorithms (EA) specifically designed to solve real parameterization optimization problems. They were presented in 1964 at the Technical University of Berlin (Germany) and developed in the 70s by Ingo Rechenberg [120] and Hans-Paul Schwefel [122]. As EAs, they use mutation, recombination, and selection applied to a population of individuals containing candidate solutions in order to evolve iteratively better and better solutions.

The first purposed ES selected just one individual (parent) to create one new (individual) offspring (see Figure 3.7). This strategy is denoted $(1 + 1) - ES$. The new population accepts the best individuals (parent or offspring) according to the Equation 3.10. Thus, less fit individuals have zero chance of survival. In $(1+1) - ES$, the new individual is generated by the next equation:

$$y_i^g \leftarrow x_i^g + N(0, \sigma)$$

where $N(0, \sigma)$ is a vector of random numbers with a Gaussian distribution with mean zero and standard deviation $\sigma$. Algorithm 3 shows the whole process of $(1 + 1) - ES$.

$$x_i^{g+1} \leftarrow \begin{cases} y_i^g & \text{si } fitness(y_i^g) \leq fitness(x_i^g), \\ x_i^g & \text{en otro caso.} \end{cases}$$

**Algorithm 3** Evolution Strategies (ES)

1: $s^0 \leftarrow \text{initializePopulation}()$
2: $\text{evaluar}(s^0)$
3: while $g < \text{maxGenerations or stopCondition}$ do
4: \hspace{1em} $s^g \leftarrow \text{mutation}(s^g)$
5: \hspace{1em} $\text{evaluation}(s^g)$
6: \hspace{1em} if $\text{fitness}(s^g)$ is better than $\text{fitness}(s^g)$ then
7: \hspace{2em} $s^g \leftarrow s^g$
8: \hspace{1em} end if
9: end while

However, the mutation by itself does not have the ability to combine the genetic information of the best individuals, so it is also necessary to introduce crossover operators. Thus, the first type of a *multimembered* ES was proposed, the $(\mu+1) - ES$ or *steady-state* ES. There are $\mu$ parents at a time. Two of them are chosen at random.
and recombined to give life to an offspring, which also underlies mutation. The selection resembles extinction of the worst, may it be the offspring or one of the parents, thus keeping constant the population size.

Subsequently, Schwefel proposed the use of two multimembered strategies called $(\mu + \lambda) - ES$ and $(\mu, \lambda) - ES$ where $\mu$ indicates the population size and $\lambda$ the offspring size [102]:

- in the $(\mu + \lambda) - ES$, not only one offspring is created at a time or in a generation, but $\lambda \geq 1$ descendants, and, to keep the population size constant, the $\lambda$ worst out of all $\mu + \lambda$ individuals are discarded;

- in the $(\mu, \lambda) - ES$, the selection takes place among the $\lambda$ offspring only, whereas their parents are forgotten no matter how good or bad their fitness was compared to that of the new generation. Obviously, this strategy relies on a birth surplus, i.e., on $\lambda > \mu$ in a strict Darwinian sense of natural selection.

Since the GAs described in Section 3.3.2, the ESs belong to EAs. Although they belong to the same family of algorithms there are some important differences between them [123]:

- ESs are applied to real parametric optimization problems, but GAs are mostly applied to optimization of attributes.

- The key operator of the GA is the recombination, however the most important operator in ES is the mutation.
3.3. ALGORITHMS USED IN THIS WORK

- By default, the replacement in the GA is stochastic, in the ES is deterministic.
- GAs are global search mechanisms, that is, they have trouble on fine adjustment, just the opposite happens with ES.

The application of evolutionary strategies is closely linked to its nature of being a tool for parametric optimization of functions. However, they also have been applied to circuit partitioning problems [62], design of materials [119] or neural network training [13]. It has also been used in the field of mobile ad-hoc networks using them to improve the performance of AODV routing protocol [77].

3.3.4 Particle Swarm Optimization (PSO)

Particle Swarm Optimization algorithms are inspired by the social behavior of bird flocking and fish schooling. More precisely, PSO is a parallel evolutionary computation technique that provides a collaborative population-based search model (see Algorithm 4). Individuals of the population called particles fly around in a multidimensional search space. During flight, each particle adjusts its position according to its own experience and according to the experience of a neighboring particle, moving toward the best position encountered by itself or its neighbors. Thus, the PSO system combines local search methods (through self-experience) with global search methods (through neighboring experience), attempting to balance exploration and exploitation [102].

Generally, a particle \( p_i \) is composed by three vectors and two fitness values:

- \( x^i = (x_{i1}, x_{i2}, ..., x_{in}) \) and \( pBest^i = (p_{i1}, p_{i2}, ..., p_{in}) \) store its current position and the best solution of the particle, respectively.
- \( v^i = (v_{i1}, v_{i2}, ..., v_{in}) \) is the gradient vector (direction) which defines the next movement of the particle \( p_i \).
- The two fitness values are the current one (\( \text{fitness}_x \)) and the value of the best local solution found so far (\( \text{fitness}_{pBest} \)).

In this algorithm, each particle position \( x^i \) is updated each generation \( g \) by means of the Equation 3.11.

\[
x^i_{g+1} \leftarrow x^i_g + v^i_{g+1}
\] (3.11)
where factor $v_{g+1}^i$ is the velocity of the particle and is given by

$$v_{g+1}^i \leftarrow w \cdot v_g^i + \varphi_1 \cdot (p_g^i - x_g^i) + \varphi_2 \cdot (b_g - x_g^i)$$  \hspace{1cm} (3.12)$$

In this formula, $p_g^i$ is the best solution that the particle $i$ has stored so far, $b_g$ is the best particle (also known as the leader) that the entire swarm has ever created, and $w$ is the inertia weight of the particle $g$ it controls the trade-off between global and local experience. Finally, $\varphi_1$ (cognitive component) and $\varphi_2$ (social component) are specific parameters which control the relative effect of the personal and global best particles ($\varphi_1 = \varphi_2 = 2 \cdot UN(0,1)$).

Algorithm 4 describes the pseudo-code of PSO. The algorithm starts by initializing the swarm (Line 1), which includes both the positions and velocities of the particles. The corresponding $p_g^i$ of each particle is randomly initialized, as well as the leader $b$ (Line 2). Then, during a maximum number of iterations, each particle flies through the search space updating its velocity and position (Lines 5 and 6), it is then evaluated (Line 7), and its $p_g^i$ is also calculated (Line 8). At the end of each iteration, the leader $b$ is updated.

**Algorithm 4 Pseudocode of PSO**

1: initializeSwarm()
2: locateLeader($b$)
3: while $g < \text{maxGenerations or stopCondition()}$ do
4:    for each particle $x_g^i$ do
5:        updateVelocity($v_g^i$) // Equation 3.12
6:        updatePosition($x_g^i$) // Equations 3.11
7:    evaluate($x_g^i$)
8:    update($p_g^i$)
9:    end for
10:   updateLeader($b_g$)
11: end while

The particles movement is showed in Figure 3.8. Where the dotted arrows represent the direction of the current velocity vectors: $v_g^{Best}$ (velocity of the best position taken by the particle), $v_g^k$ (velocity of the best particle found in the neighborhood), and $v_g^i$ (current velocity of the particle). The arrow line represents the direction taken by the particle to move from point $x_g^i$ to point $x_g^{i+1}$. The change in direction of the arrow is due to the influence of other directions (gradient) involved in the movement.
3.3. ALGORITHMS USED IN THIS WORK

PSO may be configured in different ways defining different variants of this algorithm. Kennedy in [71] defined four main variants depending on the influence of the values of the cognitive and social components ($\varphi_1$ and $\varphi_2$). They are the following ones:

- Full Model: $\varphi_1, \varphi_2 > 0$. The movement is determined by both the cognitive and social component.
- Cognition-only Model: $\varphi_1 > 0$ y $\varphi_2 = 0$. Only the cognitive component is involved in the movement.
- Social-only Model: $\varphi_1 = 0$ y $\varphi_2 > 0$. Only the social component is involved in the movement.
- Exclusive Social Model: $\varphi_1 = 0$, $\varphi_2 > 0$ y $x_i \neq g_i$. The position of the particle itself can not be the best in its environment.

According to [72] the sum of the values of the cognitive and social components of PSO ($\varphi_1$ and $\varphi_2$) should be about 4.0, and common usage is to set them 2.05 each.

This technique has been applied to problems of optimization of numerical functions [42] training neural networks [138], traveling salesman problem [125], and image processing [104]. In turn, it has been used as the basis for managing the links between nodes in VANET [61].
3.3.5 Differential Evolution (DE)

Differential Evolution is the youngest technique used to carry out the work presented here. It appears in 1994 when Kenneth Price tried to solve the Chebyshev polynomials [32]. Then, Price came up with the idea of using vector differences for perturbing the vector population. Since this seminal idea a lively discussion between Ken Price and Rainer Storm and endless ruminations and computer simulations on both parts yielded many substantial improvements which make DE the versatile and robust tool it is today [128].

DE is a stochastic population based optimization algorithm. It is employed to solve problems in continuous (real) search spaces (real parameters, real valued functions). DE optimizes a problem by maintaining a population of $N$ vectors $v^g_i$ (where $g$ is the generation number and $i \in [1, N]$) that represents the candidate solutions. The creation of new candidate solutions is carried out by combining existing vectors according to its simple formulae of differential-crossover and differential-mutation, and then keeping whichever candidate solution has the best fitness on the optimization problem at hand.

The differential-mutation generates new individuals by adding the weighted difference of two of the vectors of the population to the third, following the Equation 3.13:

$$w_i^{g+1} ← v_i^g + \mu \cdot (v_{r2}^g - v_{r3}^g). \quad (3.13)$$

Where $r1, r2, r3 \in \{1, 2, \ldots, i - 1, i + 1, \ldots, N\}$ are random integers mutually different, and also different from the index $i$, the mutation constant $\mu > 0$ stands for the amplification of the difference between the individuals $v_{r2}^g$ and $v_{r3}^g$, and it avoids the stagnation of the search process.

In order to increase even more the diversity in the population, each mutated individual undergoes a crossover operation with the target individual $v_{g}^i$, by means of which a trial individual $u_{g+1}^i$ is generated. A randomly chosen position is taken from the mutant individual to prevent that the trial individual replicates the target individual.

$$u_{g+1}^i(j) \left\{ 
\begin{array}{ll}
w_{g+1}^i(j) & \text{if } r(j) \leq Cr \text{ or } j = j_r, \\
v_i^j & \text{otherwise}.
\end{array}
\right. \quad (3.14)$$
3.3. ALGORITHMS USED IN THIS WORK

As shown in Equation 3.14, the crossover operator randomly chooses a uniformly
distributed integer value \( j \), and a random real number \( r \in (0, 1) \), also uniformly
distributed for each component \( j \) of the trial individual \( u_{g+1}^j \). Then, the crossover
probability \( C_r \) and \( r \) are compared just like \( j \) and \( j_r \). If \( r \) is less than or equal
than \( C_r \) (or \( j \) is equal to \( j_r \)) then we select the \( j \)th element of the mutant individual
to be allocated in the \( j \)th element of the trial individual \( u_{g+1}^i \). Otherwise, the \( j \)th
element of the target individual \( v_g^i \) becomes the \( j \)th element of the trial individual.
Finally, a selection operator decides the acceptance of the trial individual for the next
generation if and only if it yields a reduction in the value of the evaluation function
(also called fitness function \( f() \)), as shown by the following Equation (3.15):

\[
v_{g+1}^i \left\{ \begin{array}{ll}
u_{g+1}^i & \text{if } f(u_{g+1}^i) \leq f(v_g^i), \\
v_i^j & \text{otherwise.}
\end{array} \right.
\]  (3.15)

Algorithm 5 shows the pseudo-code of DE. After initializing the population
(Line 1), the individuals evolve during a number of generations (maxGenerations).
Each individual is then mutated (Line 5) and recombined (Line 6). The new indi-
vidual is selected (or not) following the operation of Equation 3.15 (Lines 7 and 8).

Algorithm 5 Pseudocode of DE

1: initializePopulation()
2: while \( g < \text{maxGenerations} \) or stopCondition() do
3:   for each individual \( v_g^i \) do
4:     choose mutually different\( (r_1, r_2, r_3) \)
5:     \( w_{g+1}^i \leftarrow \text{mutation}(v_{g}^{r_1}, v_{g}^{r_2}, v_{g}^{r_3}, \mu) \)
6:     \( u_{g+1}^i \leftarrow \text{crossover}(v_g^i, w_{g+1}^i, cp) \)
7:     evaluate\( (u_{g+1}^i) \)
8:     \( v_{g+1}^i \leftarrow \text{selection}(v_g^i, u_{g+1}^i) \)
9:   end for
10: end while

DE has been applied to various classical optimization problems with known
global optimum[113], also problems of industrial processes optimization, task al-
location [10], and to the design of wireless networks[114] or other problems such as
the training of neural networks [2].