Fundamentals and Practice of Evolutionary Optimization

Carlos A. Coello Coello
Raquel Hernández Gómez
Luis Miguel Antonio
CINVESTAV-IPN
Evolutionary Computation Group
Department of Computer Science
Av. IPN No. 2508, Col. San Pedro Zacatenco
Mexico City, México 07360
Emails: ccoello@cs.cinvestav.mx
rhernandez@computacion.cs.cinvestav.mx
verdepalmera@hotmail.com

September 11, 2017

Abstract

In this chapter, we describe the basics of evolutionary algorithms and their use in optimization. First, some generalities about evolutionary algorithms are provided, including a brief description of the components of the four main types of algorithms considered for the purposes of this chapter: (1) genetic algorithms, (2) evolution strategies, (3) evolutionary programming and (4) differential evolution. For the sake of brevity, this chapter only focuses on the use of evolutionary algorithms in numerical optimization. The discussion includes the use of evolutionary algorithms in both single-objective and multi-objective optimization and includes aspects such as their variation operators, selection mechanism and some of their applications.

Keywords: optimization, evolutionary algorithms, genetic algorithms, differential evolution, evolution strategies, evolutionary programming.

1 Introduction

Evolutionary algorithms (EAs) are stochastic search techniques inspired on the "survival of the fittest" principle from Darwin's evolutionary theory [1]. EAs are considered *metaheuristics*, because they operate as top-level general strategies which guide other lower-level heuristics. A heuristic is a rule or set of rules that are used to search in a more efficient way for (sub-)optimal

solutions to a (normally complex) problem. EAs are stochastic, because they rely on the use of random numbers to decide the direction in which they will move in the search space.

Although the idea of using evolution as an inspiration for solving problems can be traced as long back as the 1930s [2], it was until the 1960s when the three main techniques based on this notion were developed: the genetic algorithm (GA) [3], evolution strategies (ES) [4] and evolutionary programming (EP) [5]. The development of each of these techniques had a different motivation. GAs were meant to be used for machine learning and EP targeted the solution of prediction problems. In fact, only the development of ESs was motivated by the solution of complex optimization problems. However, over the years, both GAs and EP were also adopted as optimizers. Today, GAs are, with no doubt, the most popular EA used for optimization. GAs can be used for both combinatorial and continuous optimization problems, but their use is mainly recommended for problems having mixed variables (e.g., problems in which some decision variables are expressed as real numbers and others are expressed as integers or binary numbers) because in that domain they present important advantages with respect to most of the other existing optimizers currently available.

In 1995, Kenneth Price introduced another type of EA which has a stronger resemblance with a mathematical programming technique: differential evolution (DE) [6]. DE has become a very popular optimizer, but in its original version it was intended only for continuous domains (when all the decision variables are expressed with real numbers) [7].

In this chapter, we will briefly discuss the use of the 4 types of EAs previously indicated (GAs, ESs, EP and DE) in numerical single-objective optimization (see Section 2) and multi-objective optimization (see Section 3). Because of space constraints, combinatorial optimization is not discussed in this chapter, in spite of the fact that EAs have been used in that domain as well [8]. It is also worth noticing that many other types of bio-inspired metaheuristics exist today (see for example [9]). However, such approaches are either not EAs (e.g., artificial immune systems [10]) or they are not mainly used for optimization (e.g., genetic programming [11], which is a GA variant in which a tree encoding is used instead of a binary or real-numbers encoding).

2 Single-objective Optimization

A single-objective optimization problem can be formally defined as follows:

$$\underset{x \in \mathcal{R}^n}{\text{minimize}} \quad f(\vec{x}) \tag{1}$$

subject to:

$$g_i(\vec{x}) \leq 0 \qquad i = 1, 2, \ldots, m \tag{2}$$

$$h_i(\vec{x}) = 0 \qquad i = 1, 2, \ldots, p \tag{3}$$

where $g_i, h_j : \mathbb{R}^n \rightarrow \mathbb{R}, i = 1, ..., m, j = 1, ..., p$ are the constraint functions of the problem and $\vec{x} = [x_1, x_2, \ldots, x_n]^T$ is the vector of decision variables. $g_i(\vec{x})$ are called *inequality constraints* and $h_i(\vec{x})$ are called *equality constraints*. It is also posible to have an *unconstrained optimization* problem. In such case, no constraints are imposed on the minimization of $f(\vec{x})$.

Although a wide variety of mathematical programming techniques are currently available for dealing with different classes of optimization problems (e.g., linear, quadratic, etc.), the general nonlinear optimization problem remains open, since there is no single approach that can guarantee reaching the global optimum of any type of nonlinear problem [12]. This has motivated the use of alternative techniques, such as evolutionary algorithms (EAs).

2.1 Evolutionary Algorithms for Single-Objective Optimization

In general, an Evolutionary Algorithm (EA) operates as follows: First, we generate a set of randomly generated solutions to the problem that we aim to solve. This set of solutions is called *population*, and each element in the set is called *an individual*. Each individual is evaluated in terms of its fitness value (a measure that will tell us how good is an individual with respect to the rest of the population) such that the fittest individuals are selected as parents. Such parents are then recombined (in pairs) using an operator called *crossover* to give rise to a new set of individuals (their *offspring*). This new population is subject to another operator called *mutation*, which produces small (random) changes in each individual. After applying mutation, the cycle is repeated, since we have to determine again the fitness values of this new (mutated) population. This process is repeated until reaching a certain pre-defined stopping criterion (e.g., a maximum number of iterations (or *generations*)).

The main components of an EA are the following [13]:

Representation: It refers to the way in which we encode the decision variables of the problem that we aim to solve. All the decision variables are concatenated to form an *individual*, which will be evolved with the aim of obtaining an optimal (or at least sub-optimal) solution to the problem at hand.

Fitness function: It is used to guide the search and it normally corresponds to the original objective function of the problem to be solved (perhaps in normalized form).

Parent selection: This is a mechanism that allows us to select the pairs of parents that will be recombined (the fittest are normally selected). Two parents normally produce two offspring which are meant to be better (in terms of fitness) than their parents.

Variation operators: It refers to two main operators: crossover and mutation. Crossover combines segments of two parents to produce offspring. Mutation changes a few positions in an individual to ensure a greater diversity and to avoid getting trapped in local optima.

Survivor selection mechanism: It refers to the mechanism adopted to ensure that good (or fitter) individuals survive into future generations.

Next, we will briefly describe each of these components for the four EAs indicated in the introduction: genetic algorithms, evolution strategies, evolutionary programming and differential evolution.

2.2 Genetic Algorithms

The basic Genetic Algorithm (GA) adopts binary encoding, since this is a universal representation that can encode any type of decision variables [14]. However, other encodings are also possible (e.g., real-numbers, integer, matrix, etc. [15]). Using binary encoding, the set of bits corresponding to a single decision variable is called *gene* and each bit within a gene is called an *allele*. The concatenation of all the decision variables is called *chromosome*. Traditionally, each individual in the population of a GA contains a single chromosome. However, it is also possible to use multiple chromosomes per individual [14].

In a GA, the crossover operator leads the search process and, therefore, has the main role within the algorithm. As any other EA, a GA starts with a randomly generated set of individuals in which each allele is generated in such a way that both 0 and 1 have a 50% probability of appearing. Then, fitness is computed for each of the individuals in the population and pairs of individuals are selected for recombination (based on their fitness values). The selected individuals (i.e., the parents) are recombined in pairs using a certain probability (this is a parameter defined by the user) which is normally high (between 60% and 100%). Each pair of individuals generates two offspring upon the use of a crossover operator. The offspring produced by the crossover operator are later mutated. The mutation operator is a secondary operator, which means that it is applied with a lower probability (normally between 1% and 10%) than crossover. The offspring replace their

```
1  Begin
2      G=0
3      Create a randomly generated initial population C(G)
4      For G=1 to GMAX (maximum number of generations) Do
5          Evaluate fitness of all the individuals in C(G)
6          Select best-fit individuals (parents) from C(G) for reproduction
7          Perform crossover and mutation to create an offspring population C'(G)
8          The offspring population becomes the new population C(G)
9          G = G + 1
10     End For
11 End
```

Figure 1: Simple Genetic Algorithm

parents and the whole process is repeated for a certain number of iterations (called *generations*). The selection process in a GA is normally based on the fitness contribution of each individual to the total fitness of the population. This is called *probabilistic selection* and aims to give a fair treatment to individuals having low fitness values, so that their probability of being selected, although low, is greater than zero. The full pseudo-code of a simple genetic algorithm is shown in Figure 1.

GAs tend to have a poor performance in problems in which there is a very high interaction among the decision variables (this is called *epistasis*), and offer great advantages in optimization problems having mixed variables (e.g., when some decision variables are real numbers, others are integers and so on).

Next, we will briefly discuss the main variation operators (crossover and mutation) available for binary and real-numbers encoding, as well as some of the most popular parent selection and survival selection schemes available for GAs. Additionally, we will provide a short discussion on constraint-handling techniques available for GAs as well as some of their applications.

2.2.1 Variation Operators

Several crossover operators exist for each type of encoding. If binary encoding is adopted, the most popular crossover operators are the following:

One-point crossover: In this case, a single point is randomly chosen along a chromosome. The first offspring is generated by copying the bits from the first parent to the crossover point and the bits from the crossover point to the end of the chromosome from the second parent. The second offspring is generated in an analogous way. This is illustrated

in Figure 2. This crossover has a strong bias against holding together genes that are located at opposite ends of the representation

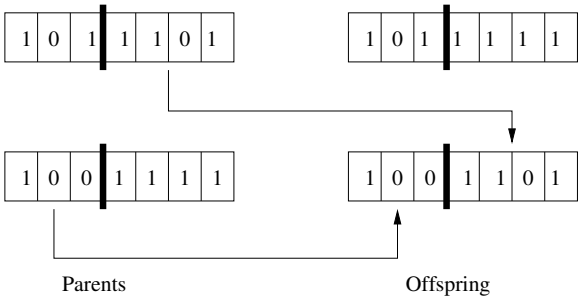

Figure 2: One point crossover.

Two-points crossover: In this case, two crossover points are randomly produced. The first offspring is produced by using the extremes of parent 1 and the middle part from parent 2. The second offspring is generated in an analogous way.

Two points crossover is illustrated in Figure 3. This crossover tends to maintain together genes that are located close to each other in the cromosome.

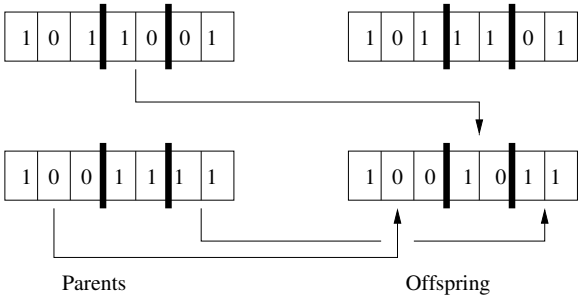

Figure 3: Two points crossover.

Uniform crossover: Introduced in [16], each bit of each offspring is selected with a certain probability from each of its two parents. For example, if a crossover rate of 50% is adopted, half of the bits of each offspring will come from each parent. Uniform crossover is illustrated in Figure 4. This crossover does not allow to transmit a large number of coadapted genes from parents to the offspring.

This kind of crossover operators have the disadvantage that only mutation can insert new values into the population, since these recombinations only create new combinations of existing values.

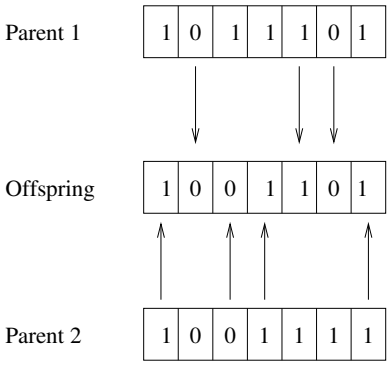

Figure 4: Uniform crossover.

The mutation operator adopted with binary encoding is very simple since it consists in an inversion of a bit (e.g., a zero becomes one and viceversa). Mutation is normally applied with a low probability to each allele of each offspring in the population.

In spite of the fact that binary encoding can be used to encode any sort of decision variable, in practice, some problems are solved in an more efficient manner with different types of encodings. For example, if the solution to the optimization problem is a permutation of integers, the use of an integer encoding may be more efficient and effective than the use of a binary encoding. However, the use of integer encoding requires different variation operators.

In evolutionary optimization, it is very common to use real-numbers encoding (i.e., to have chromosomes that consist of a vector of real numbers). This sort of encoding can be very effective for solving numerical optimization problems, but a different type of crossover operator is evidently required. Some possible crossover operators for real-numbers encoding are the following:

Arithmetic crossover: This operator [15] is a multi-parent recombination technique which uses a weighted average approach from n parents to create offspring. New solutions are created using

$$y = \sum_{j=1}^{n} \varphi_j x_j, \quad where \quad \sum_{j=1}^{n} \varphi_j = 1 \tag{4}$$

With this kind of operator, recombination is capable of creating new gene material, however, it has the disadvantage that as an averaging process is adopted for the computation of the new genes, the range of the allele values in the population for each gene is reduced.

Blend crossover: The blend crossover operator (BLX-α) [17] is an arithmetic crossover operator which creates new solutions using the follow-

ing expression

$$y = (1-\beta)x_1 + \beta x_2, \quad with \quad \beta = (1+2\alpha)u - \alpha \tag{5}$$

Where u is a random number sampled uniformly from $[0, 1]$. An $\alpha = 0.5$ value is usually adopted. BLX-α creates new solutions whose location depends on the distance that the parents are away from one another. That is to say, when the distance between parents is small, so is the distance between the new solution and its parents. This crossover has the characteristic of creating new material without restricting the range of the solutions.

Simulated binary crossover: This operator (abbreviated as SBX) [18] creates new solutions which are symmetric to their parents by simulating the way one-point crossover works on a binary encoding. SBX generates offspring using:

$$y_{1j} = 0.5[(1+\gamma_j)x_{1j} + (1-\gamma_j)x_{2j}] \tag{6}$$
$$y_{2j} = 0.5[(1-\gamma_j)x_{1j} + (1+\gamma_j)x_{2j}] \tag{7}$$

where

$$\gamma_j = \begin{cases} (2r_j)^{\frac{1}{\mu+1}} & \text{if} \quad r_j \leq 0.5 \\ (\frac{1}{2(1-r_j)})^{\frac{1}{\mu+1}} & \text{otherwise} \end{cases} \tag{8}$$

where r_j is a random number in the range $[0, 1]$ generated from a uniform distribution and μ is a distribution index controlling the closeness that the offspring have from their parents. Normally, $\mu = 1$.

This kind of crossover prevents bias towards any of the parents. μ value controls the closeness of the new solutions to their parents: the smaller the value of μ, the farther away the offspring will be from their parents.

The use of real-numbers encoding also requires specialized mutation operators. For example:

Non-Uniform Mutation: It was proposed in [15] and it operates as follows: Given:

$\vec{P} = [v_1, ..., v_m]$ the mutated individual will be: $\vec{P'} = [v_1, ..., v'_k, ..., v_m]$
where: $v'_k = \begin{cases} v_k + \Delta(t, UB - v_k) & \text{if } R > 0.5 \\ v_k - \Delta(t, v_k - LB) & \text{if } R <= 0.5 \end{cases}$

and the decision variable v_k is in the range $[LB, UB]$[1]. R is random number in the range [0,1]. $\Delta(t, y)$ returns a value in the range $[0, y]$ such that the probability that $\Delta(t, y)$ is close to zero is increased as t (current generation) increases. Michalewicz suggests using: $\Delta(t, y) = y * (1 - r^{(1-\frac{t}{T})^b})$ where: r is a randomly generated real number in the range $[0, 1]$, T is the maximum number of generations and b is a parameter that describes the non-uniformity degree adopted by the mutation operator (Michalewicz [15] suggests using $b = 5$).

This operator explores in a more global manner the search space at the beginning (when t is small) and in a more local (focalized) way towards the end of the search process.

Uniform Mutation: Given: $\vec{P} = [v_1, ..., v'_k, ..., v_m]$ the mutated individual will be: $\vec{P'} = [v_1, ..., v'_k, ..., v_m]$ where: $v'_k = rnd(LB, UB)$. In this case, a uniform distribution is adopted and [LB, UB] define the lower and upper bounds of the decision variable v'_k.

This mutation operator is usually adopted with a positionwise mutation probability and its purpose is to introduce new values into the population.

Parameter-Based Mutation: Proposed by Deb [18]. Assuming that the parent y is bounded ($y \in [y_l, y_u]$), the operator procedure is the following:

1. Generate a random number u in the range [0,1].
2. Compute:

$$\delta_q = \begin{cases} \left[2u + (1-2u)(1-\delta)^{\eta_m+1}\right]^{\frac{1}{\eta_m+1}} - 1 & \text{if } u \leq 0.5 \\ 1 - \left[2(1-u) + 2(u-0.5)(1-\delta)^{\eta_m+1}\right]^{\frac{1}{\eta_m+1}} & \text{otherwise} \end{cases} \tag{9}$$

where $\delta = \min\left[(y - y_l), (y_u - y)\right]/(y_u - y_l)$.

The parameter η_m is the distribution index for the mutation operator and it takes any non-negative value. Deb suggests using:

$$\eta_m = 100 + t \quad (t = \text{current generation})$$

3. The value of the mutated position is determined by using:

$$y' = y + \delta_q \Delta_{max}$$

where Δ_{max} is the maximum allowable perturbation:

[1] $[LB, UB]$ define the lower and upper bounds of the decision variable V_k.

$$\Delta_{max} = y_u - y_l$$

considering that:

$$y \in [y_l, y_u]$$

The aim of this operator is to generate a solution c in the neighborhood of a parent solution y.

2.2.2 Parent Selection Mechanism

As indicated before, an usual way of implementing parent selection in a GA is by adopting a **fitness proportionate selection** scheme (FPS). This operator choses an individual f_i for recombination with a probability of $f_i / \sum_{j=1}^{n} f_j$, where n is the number of individuals in the population. Therefore, the probability of a solution to be selected depends on the absolute fitness value of the current individual compared to the absolute fitness values of the entire population.

An alternative parent selection mechanism is **Ranking selection** [19]. The main motivation of this operator is to preserve a constant selection pressure during the search process. For this sake, ranking selection sorts the current population of parents based on their fitness values (in decreasing order). Then, it allocates a selection probability to each individual based on their rank instead of using their current fitness value. The mapping from this rank to a selection probability can be performed in several ways (e.g., in a linearly or exponentially decreasing way).

One of the most popular methods for parent selection is **Tournament selection**. This method uses an ordering relation to compare individuals (normally in pairs, using the so-called binary tournament selection scheme). Each group of individuals "compete" in a tournament in which the only winner is the one with the highest fitness value from the group. It is required to perform a sufficient number of tournaments to produce a number of parents that is equal to the population size.

2.2.3 Survivor Selection Mechanism

In traditional GAs the population of parents is totally replaced by the population of offspring. It is worth noting, however, that due to theoretical reasons, GAs normally adopt an additional operator called *elitism* [20], by which the best individual at each generation is retained intact in the next generation.

Other survival selection schemes are evidently possible, such as the *plus* selection scheme by which the best m individuals (m is the population size) are selected from the union of parents and offspring (this union will normally be of size $2 \times m$). This scheme is implicitly elitist, because the best individual

from the union of both populations will always survive. There is evidence indicating that the use of plus selection favors the performance of GAs that use real-numbers encoding for solving numerical optimization problems [21].

2.2.4 Constraint-Handling Techniques

As optimizers, GAs are unconstrained search techniques. However, it is clear that most real-world applications have constraints. Therefore, a constraint-handling mechanism is required to allow GAs to properly solve constrained optimization problems. Several types of constraints are possible. For example, decision variables may allow only a certain range of values (e.g., $-5 \leq x_1 \leq 200$). These are called *bound constraints*, and can be handled directly when encoding the decision variables. However, equality and inequality constraints imposed on the definition of the optimization problem require a specific mechanism that must be added to the GA.

In the early days of EAs, penalty functions [22] were the most popular constraint-handling technique. When using a penalty function, a constrained problem is transformed into an unconstrained one in which the fitness value is "punished" (or penalized) when a solution is infeasible (i.e., when it violates one or more constraints from the problem). The higher the violation of the constraints, the higher the penalty that is applied. Although relatively simple, penalty functions have several disadvantages. The main one is that they rely on a *penalty factor* that is used to balance the influence that the constraint violations will have in the computation of the (penalized) fitness value. In general, it is not possible to know, for an arbitrary problem, the most appropriate value of the penalty factor, and such value has a serious impact on the performance of the evolutionary optimizer. Although several types of schemes have been proposed to define a penalty factor whose value is defined (e.g., adapted based on certain measures) during the evolutionary process, in general, it is very difficult to define a good penalty factor [23]. Motivated by this disadvantage, several other (more elaborate) constraint-handling techniques for GAs have been proposed over the years [24].

2.2.5 Recent developments

Some of the most relevant research on GAs developed on the last few years has focused on automated parameter tuning, using either mechanisms such as self-adaptation [25] (in which the parameters are incorporated as additional decision variables in the chromosome so that the GA evolves them) or the so-called automatic algorithm configuration approaches [26]. Other relevant topics have been multi-objective optimization [27], large scale optimization [28] and techniques to preserve diversity [29] with the aim of avoiding premature convergence.

2.2.6 Applications

GAs have been applied to a wide variety of optimization problems, including structural optimization [30], architectural design [31], chemical engineering [32], aeronautical engineering [33] and electrical engineering [34] among many others.

2.3 Evolution Strategies

Evolution strategies [35] are based on the idea that evolution can be able to optimize itself. The first evolution strategy, known as $(1 + 1)$-ES, works in the following manner: First, a single individual, consisting of a vector of real numbers is "mutated" (i.e., a single decision variable in this individual is replaced by another one which is at a randomly generated distance from such individual). The new individual (the offspring) is compared with respect to its parent (i.e., the unmutated individual) in terms of fitness value. The individual with a better fitness value is the one that survives. A new individual is generated using:

$$\vec{x}_{t+1} = \vec{x}_t + N(0, \sigma) \tag{10}$$

where t is the current generation and N is a randomly generated number according to a Normal distribution that has a zero mean and a standard deviation σ. The most interesting part of the $(1 + 1)$-ES is that it is able to self-adapt its mutation rate by using the so-called $\frac{1}{5}$ success rule. What the rule says is that the ratio between successful mutations and the total number of mutations performed by the algorithm must be exactly $\frac{1}{5}$. If this ratio is higher than $\frac{1}{5}$, then, the standard deviation must be increased. Otherwise, it must be decreased. A mutation is considered successful when the offspring that it generates has a better fitness than its parent.

So, the $\frac{1}{5}$ success rule is expressed as:

$$\sigma(t) = \begin{cases} \sigma(t-n)/c & \text{if } p_s > 1/5 \\ \sigma(t-n) * c & \text{if } p_s < 1/5 \\ \sigma(t-n) & \text{if } p_s = 1/5 \end{cases}$$

where n is the number of decision variables of the problem t is the current generation, p_s is the relative frequency of successful mutations measured over a certain time interval (e.g., after mutating $10 \times n$ individuals) and $c = 0.817$ (this value was theoretically derived by Schwefel [36]). $\sigma(t)$ is adjusted after performing n mutations. Algorithm 5 describes a $(1 + 1)$-ES.

2.3.1 Variation Operators

In an Evolution Strategy, mutation is the main variation operator, and several forms of mutation are possible. For example, when using the **Uncorrelated Mutation with One Step Size**, the same distribution is adopted

1 **Begin**
2 t=0
3 Create a random initial population $P(t) \leftarrow \vec{x}$
4 Evaluate $f(\vec{x})$
5 Initialize archive A of successful mutations
6 **For** t=1 to MAX_GEN **Do**
7 $\vec{x}' = \vec{x} + N(0, \sigma(t))$
8 **If** $f(\vec{x}') < f(\vec{x})$ **Then**
9 $\vec{x} \leftarrow \vec{x}'$
10 store success in A
11 **Else**
12 store failure in A
13 **End If**
14 $P(t+1) \leftarrow \vec{x}$
15 **If** mod $n = 0$ **Then**
16 get $num\_successes$ and $num\_failures$ from A
17 $p_s = \frac{num\_successes}{num\_successes + num\_failures}$
18 $\sigma(t) = \begin{cases} \sigma(t-n)/c & \text{if } p_s > 1/5 \\ \sigma(t-n) * c & \text{if } p_s < 1/5 \\ \sigma(t-n) & \text{if } p_s = 1/5 \end{cases}$
19 **End If**
20 $t = t + 1$
21 **End For**
22 **End**

Figure 5: $(1+1)$-ES algorithm. Here, n is the number of decision variables of the problem, t is the current generation and $c = 0.817$. "MAX_GEN"' is a user-defined parameter representing the maximum number of generations (iterations) to be used.

to mutate the decision variables, so only one strategy parameter σ is used for all the decision variables. It is also possible to use the **Uncorrelated Mutation with n Step Sizes**, in which each set of n decision variables is extended with n step sizes, one for each decision variable.

Crossover can also be applied in evolution strategies, but this is a secondary operator, because it is normally applied with a lower probability.

The most common crossover operator involves the use of two or more parents to produce a single offspring. Evolution strategies adopt two main crossover operators: **discrete recombination** and **intermediate recombination**.

When using discrete recombination, each element of the offspring is randomly taken from one of the parents. When using intermediate recombination, each element of the offspring is computed as the average of the values of its parents.

Crossover operators in evolution strategies also differ depending on the number of parents which are recombined to produce a new solution. Two main approaches are commonly adopted. In the first scheme, which is called **local crossover**, two individuals are randomly selected from the population to act as parents. In the second scheme, known as **global crossover**, more than two individuals are randomly selected to breed. It is worth noting that evolution strategies adopt a randomly-based parent selection strategy, and not one based on fitness as done in the genetic algorithm.

2.3.2 Survivor Selection Mechanism

Rechenberg [35] introduced the use of populations in evolution strategies, with the so-called ((μ+1)-ES, in which μ parents generate a single offspring, which will replace to the worst parent from the population.

Schwefel [36] proposed the use of multiple offspring in the so-called (μ+λ)-ES and (μ,λ)-ES. In the first case ((μ+λ)-ES), the μ best individuals obtained from the union of parents and offspring are the ones who survive. In the second case ((μ,λ)-ES), the μ best offspring survive (i.e., there is a total replacement of the parents population, as done in the traditional genetic algorithm).

2.3.3 Constraint-Handling Techniques

The first versions of the evolution strategy used an approach called *death penalty*. In this approach, an infeasible solution is discarded and a new one is randomly generated. This approach, although simple, only works when the size of the feasible region is fairly large with respect to the size of the entire search space. Over the years, more elaborate constraint-handling techniques were introduced (see for example [37]). Additionally, constraint-handling techniques used with other evolutionary algorithms can also be

adopted with evolution strategies (see for example [38]).

2.3.4 Recent developments

One of the most successful versions of Evolution Strategies is the Covariance Matrix Adaptation Evolution Strategy (CMA-ES), first proposed in [39]. In CMA-ES, the adaptation of a covariance matrix is used to learn a second order model of the objective function, analogous to the use of the approximation of the inverse Hessian matrix in Quasi-Newton methods. However, in this case neither derivatives nor even the function values themselves are required by the method, since only the ranking between candidate solutions is used for learning the sample distribution. This mechanism allows CMA-ES to ensure invariance to any kind of rotation of the fitness function. Additionally, CMA-ES is capable of dealing efficiently with high-dimensional non-separable ill-conditioned objective functions.

2.3.5 Applications

Evolution strategies have been mainly used for numerical optimization in a wide variety of domains, including medicine [40], transport engineering [41], biotechnology [42] and structural optimization [43] among many others.

2.4 Evolutionary Programming

The development of Evolutionary Programming (EP) [5] was inspired on the adaptive behavior present in evolution, with the idea that evolution can be seen as a learning process that gives rise to artificial intelligence. EP models the evolutionary process at the species level and not at an individual level (as done by GAs and ESs). Therefore, EP does not consider recombination (different species cannot be recombined).

EP requires no encoding of solutions (same as ESs), so if we aim to solve an optimization problem in which de decision variables are real numbers, EP will use a vector of real numbers to represent each individual in the population. Originally, EP was used to evolve finite state machines that aimed to produce a set of outputs defined in a transition table. In this case, fitness was based on the behavioral error of individuals for the given environment.

In general, EP starts with a set of solutions that are randomly generated. Then, fitness is computed for each of these individuals. Parents are selected based of their fitness value, but normally through the use of deterministic tournaments (i.e., the best individual always wins). Selected individuals are mutated (several types of mutation operators are available) to produce the offspring that constitute the next generation. This process is repeated until reaching a certain stopping criterion. The basic EP algorithm is presented in Figure 6.

```
1  Begin
2      G=0
3      Create a random initial population C(G) ← x⃗_{i,G} ∀i, i = 1,...,NP
4      Evaluate f(x⃗_{i,G}) ∀i, i = 1,...,NP
5      For G=1 to MAX_GEN Do
6          For i=1 to NP Do
7              Create an offspring x⃗'_{i,G} by applying the mutation operator
8              Evaluate the fitness f(x⃗'_{i,G})
9              Add x⃗' to the set offspring set C'(G)
10         End For
11         Select the new population C(G+1) from C(G) ⋃ C'(G)
12         G = G + 1
13     End For
14 End
```

Figure 6: Basic EP Algorithm. "NP" and "MAX_GEN"' are user-defined parameters.

2.4.1 Variation Operators

In evolutionary programming, each parent is used to produce one or more offspring from it through the use of mutation. Mutation operators can be tailored for the problem at hand, but in general, a new individual is generated using:

$$x'_i = x_i + \phi(\sigma_i)\eta_i \tag{11}$$

where ϕ is a function that calibrates the noise η_i. Based on the features of the scaling function, there are three main variants of EP. The first is the so-called **non-adaptive** version, where the deviation step sizes are static throughout the whole evolutionary process. The second is the so-called **dynamic** version, where the changes in step sizes occur over time using ϕ as the fitness values of the individuals. The third one is the **self-adaptive** version, in which step sizes change dynamically and the best values are learnt from the decision variables of the problem.

2.4.2 Survivor Selection Mechanism

The most commonly adopted way to select individuals to survive is to select the best individuals among parents and offspring. To define what solutions are better, a relative fitness measure is adopted. This measure indicates how well an individual performs against other solutions selected from the parents and offspring, through pairwise tournament competitions, performed in a

round-robin fashion. At the end, the solutions with the greatest number of victories are chosen to pass to the next generation.

2.4.3 Constraint-Handling Techniques

There are some specific constraint-handling mechanisms that have been developed for EP. For example, the so-called *two-phase evolutionary programming* [44]. As its name indicates, this approach operates in two phases. In the first phase, a penalty function is applied. Thereafter, the best individual from phase one is used to generate the offspring for phase two, which optimize the dual Lagrangian problem. Evidently, the same constraint-handling techniques proposed for other EAs can also be used with EP (see for example [38]).

2.4.4 Recent developments

Recent work on EP have been focused on the development of new mutation operators. For example, in [45], the authors study the development of an automatic process to design a mutation operator for EP.

2.4.5 Applications

EP is the least frequently used EA, but it has been applied in several domains, including networking [46], operating systems [47] and electronics [48], among others.

2.5 Differential Evolution

Differential Evolution (DE) [6] is an EA that was originally designed for solving continuous optimization problems and which has been found to be very effective in a wide variety of problems [7]. The original version of DE adopts real-numbers encoding, since it is assumed that in the optimization problem to be solved, all the decision variables are real numbers. DE performs mutation based on the distribution of the solutions in the current population. Thus, the search directions and the possible step sizes to be used by the algorithm depend on the location of the individuals selected to calculate the mutation values.

The original proposal used a certain nomenclature to refer to the different DE variants proposed by the authors of this algorithm [6]. The most popular DE variant is known as *"DE/rand/1/bin"*, where "DE" means Differential Evolution, "rand" indicates that the individuals selected for computing the mutation values are randomly chosen, "1" is the number of pairs of solutions that are selected and "bin" indicates that a binomial recombination is used. The corresponding algorithm of this variant is presented in Figure 7.

```
1   Begin
2       G=0
3       Create a random initial population x⃗_{i,G} ∀i, i = 1,...,NP
4       Evaluate f(x⃗_{i,G}) ∀i, i = 1,...,NP
5       For G=1 to MAX_GEN Do
6           For i=1 to NP Do
7 ⇒             Select randomly r1 ≠ r2 ≠ r3 :
8 ⇒             j_rand = randint(1, D)
9 ⇒             For j=1 to D Do
10 ⇒                If (rand_j[0,1) < CR or j = j_rand) Then
11 ⇒                    u_{i,j,G+1} = x_{r3,j,G} + F(x_{r1,j,G} − x_{r2,j,G})
12 ⇒                Else
13 ⇒                    u_{i,j,G+1} = x_{i,j,G}
14 ⇒                End If
15 ⇒            End For
16              If (f(u⃗_{i,G+1}) ≤ f(x⃗_{i,G})) Then
17                  x⃗_{i,G+1} = u⃗_{i,G+1}
18              Else
19                  x⃗_{i,G+1} = x⃗_{i,G}
20              End If
21          End For
22          G = G + 1
23      End For
24  End
```

Figure 7: "DE/rand/1/bin" algorithm. randint(min,max) is a function that returns an integer number between *min* and *max*. rand$[0,1)$ is a function that returns a real number between 0 and 1. Both are based on a uniform probability distribution. "NP", "MAX_GEN", "CR" and "F" are user-defined parameters. "D" is the number of decision variables of the problem. Steps marked with arrows change depending on the DE version adopted.

The "CR" parameter controls the influence of the parent in the generation of its offspring. Higher values imply less influence of the parent. The "F" parameter scales the influence of the set of pairs of solutions selected to calculate the mutation value (one pair in the case of the algorithm in Figure 7).

It is worth noticing that an increase on either the population size or the number of pairs of solutions used to compute the mutation values, will also increase the diversity of possible movements, thus promoting a better exploration of the search space. However, this will also considerably decrease the probability of finding the correct search direction. Therefore, the balance between the population size and the number of differences determines the efficiency of the DE algorithm [49]. Besides this balance, another important factor when using this algorithm is the type of variant that is adopted. Each DE variant provides a different mechanism to compute the mutation values as well as different types of recombination operators.

Next, we will briefly describe eight DE variants taken from [50]. The main differences among these variants lie on the recombination operator adopted (see steps 9 to 15 in Figure 7) and also in the way individuals are selected to compute the mutation vector (see step 7 in Figure 7). The variants are the following:

- Four DE variants whose recombination operator is discrete, always using two individuals: the original parent and the DE mutation vector (step 11 in Figure 7). Two discrete recombination operators: binomial and exponential. The main difference between them is that for binomial recombination, each variable of the offspring is taken from one of the two parents each time, based on the "CR" parameter value. On the other hand, in the exponential recombination, each variable of the offspring is taken from the first parent until a random number exceeds the "CR" value. From that moment on, all the offspring variable values will be taken from the second parent. These variants are called: *"DE/rand/1/bin"*, *"DE/rand/1/exp"*, *"DE/best/1/-bin"* and *"DE/best/1/exp"* [51]. The "rand" variants select all the individuals to compute mutation at random and the "best" variants use the best solution in the population besides the random ones.

- Two DE variants with arithmetic recombination, which, unlike discrete recombination, is rotation invariant. These are *"DE/current-to-rand/1"* and variant *"DE/current-to-best/1"* [51]. The only difference between them is that the first selects the individuals for mutation at random and the second one uses the best solution in the population besides random solutions.

- *"DE/rand/2/dir"* [49], which incorporates objective function information to the mutation and recombination operators. The aim of this ap-

proach is to guide the search to promising areas faster than traditional DE. Their authors argue that the best results are obtained when the number of pairs of solutions is two [49].

- Finally, a DE variant with a combined discrete-arithmetic recombination, which is called *"DE/current-to-rand/1/bin"* [51].

2.5.1 Constraint-Handling Techniques

There have been some proposals of constraint-handling mechanisms for DE. For example, in the approach proposed in [52], an infeasible solution is assisted so that it can quickly move towards the feasible region by making a consensus among the currently violated constraints. However, it should be clear that the same constraint-handling techniques proposed for other EAs can also be used with DE (see for example [38]).

2.5.2 Recent developments

Recent research on DE has focused on topics such as self-adaptation (see for example [53], where a history-based adaptation of the control parameters F and CR is proposed) and on the effects of different recombination operators (see for example [54]).

2.5.3 Applications

DE has been widely used in continuous optimization. Some of the domains in which it has been applied include thermal engineering [55], geophysics [56] and image processing [57], among many others.

2.6 Parameters settings

To create an EA instance requires choosing values for its respective parameters. Such values determine the performance of an EA and, therefore, the quality of the obtained solutions. The common practice is to define parameter values before the execution of the algorithm and to maintain them fixed during the run. Although some rules-of-thumb are normally adopted for setting the parameters of an EA, this sort of approach is fully empirical and based on some limited experimentation with a handful of values. The limitations of this sort of approach should be very evident, since it is very time consuming to try all different combinations of parameter values (for example, testing five different values for five parameters would take $5^5 = 3125$ setups). Besides, there exist theoretical arguments which state that for each specific problem, EAs may require specific parameters settings and that generally it is not possible to know *a priori* such settings [13]. For this reason, recent approaches to parameter tuning consider the development of tuning methods based on search algorithms (tuning methods), which aim to optimize

two main aspects: obtained solution quality and algorithm speed [58, 59]. This is achieved by modeling the problem of parameter tuning as an optimization problem which consists on finding the parameter vectors with the maximum utility. So in general, the solution of a tuning problem depends on the problem to be solved, the adopted EA and the utility function (which defines the way in which the algorithm's quality is measured).

Figure 8 shows the most widely adopted taxonomy for parameters settings in EAs [60], according to the way in which the parameter tuning is done and the time in which parameters are defined.

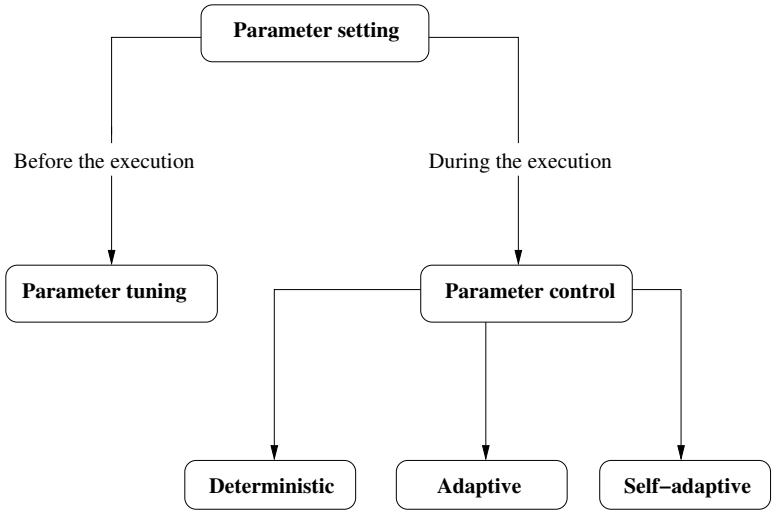

Figure 8: Taxonomy of parameters settings approaches used with EAs

For the case of parameter control, the different available methods are the following:

Deterministic parameter control: Consists on a definition of parameters given by certain rules adopted by the user, without taking into account any feedback from the search process. Rules are activated at specified intervals during the execution.

Adaptive parameter control: This method uses feedback from the search process to determine the change of the parameters. However, the mechanism is not part of the usual evolutionary process but an external mechanism which provides control parameters. Usually, such mechanism is based on the quality of solutions obtained by the different parameters, so that the mechanism can distinguish between the merits of each of the parameters settings.

Self-adaptive parameter control: This parameter control mechanism encodes parameters into the individuals, so that mutation and recombination operators can act over them. In this way, better parameter

values lead to better individuals, which in turn have more chances to survive, then produce offspring and therefore propagate these better parameter values to further stages of the optimization process.

Nowadays, parameter tuning has become an important research area which has been extended to full automatic algorithm configuration approaches [59, 13, 61, 26].

3 Multi-Objective Optimization

Many real-world problems have two or more (often conflicting) objectives that we aim to optimize at the same time. These are the so-called multi-objective optimization problems (MOPs) and they rarely have a single solution that simultaneously optimizes all the objectives. Instead, when solving a MOP we aim to find the best possible trade-offs among all the objectives (i.e., solutions in which it is not possible improving one objective without worsening another).

Formally, a MOP is defined by a decision space $\mathcal{X} \subset \mathbb{R}^n$, an objective space $\mathcal{Z} \subset \mathbb{R}^k$, and k objective functions $f_1, f_2, \ldots, f_k$ to be minimized. Each decision vector $\mathbf{x} \in \mathcal{X}$ is related to an objective vector $\mathbf{f} = (f_1(\mathbf{x}), \ldots, f_k(\mathbf{x})) \in \mathcal{Z}$, where $\mathcal{X}$ is usually restricted by bounds on the decision variables $x_i \in [x_i^l, x_i^u]$ for $i = 1, 2, \ldots, n$, inequality constraints $g_i(\mathbf{x}) \leq 0$ for $i = 1, 2, \ldots, m$, and equality constraints $h_j(\mathbf{x}) = 0$ for $j = 1, 2, \ldots, p$.

When dealing with MOPs, it is not possible to compare directly two solutions $\mathbf{x}, \mathbf{y} \in \mathcal{X}$ as in single-objective optimization. Instead, the Pareto dominance relation is normally used: it is said that $\mathbf{x}$ dominates $\mathbf{y}$ ($\mathbf{x} \prec \mathbf{y}$) if $\mathbf{x}$ is at least as good as $\mathbf{y}$ in all objectives ($\forall i \in \{1, \ldots, k\}\ f_i(\mathbf{x}) \leq f_i(\mathbf{y})$) and better in at least one objective ($\exists j \in \{1, \ldots, k\}$, $f_j(\mathbf{x}) < f_j(\mathbf{y})$). Thus, three scenarios may occur, either $\mathbf{x} \prec \mathbf{y}$, $\mathbf{y} \prec \mathbf{x}$ or neither of both, i.e., they are *non-dominated* to each other. This binary relation induces a strict partial order on $\mathcal{X}$. Thus, the solution to a MOP consists of finding the optimal set of non-dominated decision vectors $\{\mathbf{x}^* \in \mathcal{X} \colon \nexists \mathbf{x} \in \mathcal{X}, \mathbf{x} \prec \mathbf{x}^*\}$, which cannot be improved in any objective without worsening at least another. Such set is known as the *Pareto optimal set* and its corresponding image in $\mathcal{Z}$ is called the *Pareto optimal front*.

Multi-objective evolutionary algorithms (MOEAs) differ from single-objective EAs in the way both the parent and the survival selection processes are performed. In MOEAs, we aim the following:

1. Convergence to the Pareto optimal front,
2. A uniform distribution of solutions along the Pareto front and
3. A good spread of solutions, such that all of the Pareto front is covered.

The last two goals are closely related, and they are related to the *diversity* of the population. On the other hand, the variation operators of single-objective EAs might be directly implemented on MOEAs.

MOEAs offer several advantages with respect to mathematical programming techniques. The main ones are that MOEAs require little specific domain information, they are less susceptible to the shape and continuity of the Pareto front of the problem to be solved, and they can generate several elements of the Pareto optimal set in a single execution. In contrast, mathematical programming techniques normally require that the objective functions and the constraints of a MOP are differentiable, some of them are very susceptible to the shape of the Pareto front (e.g., some of them do not properly work when dealing with disconnected Pareto fronts) and most of them generate a single element of the Pareto optimal set per execution [62].

As in single-objective optimization, elitism is a very important operator in MOEAs due to theoretical reasons, since it is required to guarantee convergence to the Pareto optimal set [63]. Elitism operates by retaining a maximum number of non-dominated solutions during the evolutionary process. Elitism is implemented either through the survival selection mechanism $(\mu + \lambda)$ or by using an external archive, which is a data structure that resides in main memory and whose aim is to retain as many non-dominated solutions as possible.

In what follows, a brief historical review of the most representative MOEAs is presented. The techniques described next are classified into four categories: (1) non-elitist non-Pareto-based, (b) non-elitist Pareto-based methods, (3) elitist Pareto-based approaches and (4) elitist non-Pareto-based algorithms. Each of these categories is briefly described next.

3.1 Non-Elitist Non-Pareto-based Methods

Within this group, we consider the oldest MOEAs reported in the literature. These algorithms are non-elitist and do not adopt Pareto dominance in their selection mechanism. Though they are straightforward and computationally efficient, they have no explicit mechanism to maintain diversity.

3.1.1 Lexicographic ordering

This is a very old mathematical programming technique in which the objectives are ranked based on their importance (this is defined by the user). At each iteration, the most important objective in the sequence is minimized, while the others are transformed into equality constraints. Each constraint keeps an objective at a certain level of satisfaction, which corresponds to its best value found so far. In the early days of MOEAs, some researchers proposed to couple lexicographic ordering with evolutionary algorithms. For example, in [64], this method was coupled to a GA. In this case, the parent

selection is performed at random, and the survival selection mechanism is based on binary tournaments adopting two approaches. In one, the comparison is deterministic using as a criterion the current objective, and when there is a tie, the next objective in the ranking is chosen. In the second approach, at each comparison, a random objective is selected by a predefined frequency.

The main disadvantage of this approach is that the performance depends on the ordering/frequency imposed on the objectives, having the undesirable consequence of making the population converge to a particular region of the Pareto optimal front. Additionally, this sort of approach is not appropriate for problems having more than two objectives.

3.1.2 Linear Aggregating Functions

This is the oldest mathematical programming used for solving MOPs, and it has also been used many times with evolutionary algorithms. The basic idea of this approach is to combine all the objectives into a single scalar value: $\sum_{i=1}^{k} w_i f_i(\mathbf{x})$. This scalar value is used as the fitness of an evolutionary algorithm. It is worth noticing that when using this approach, the objectives need to be properly scaled. Normally, it is assumed that $\sum_{i=1}^{k} w_i = 1$. Several authors have used linear aggregating functions with GAs (see for example [65]).

The main drawbacks of this method are related to the difficulties associated with the definition of the weight values and to the fact that linear aggregating functions are unable to generate non-convex portions of the Pareto front [66].

3.1.3 VEGA

The Vector Evaluated Genetic Algorithm (VEGA), proposed by Schaffer [67], is considered the first actual implementation of a MOEA. The parent selection in VEGA is performed as follows. First, the population is randomly split into k subpopulations of equal size. Each subpopulation is associated with a different objective, and the individuals belonging to this partition are ranked according to their performance in such objective. Next, the mating pool is created using proportionate selection [14]. Finally, the population is combined to apply the variation operators. The main limitations of VEGA are that it suffers from some bias towards extreme points. Also, when proportionate selection is used, VEGA's selection mechanism behaves similarly to a linear aggregating function, which implies that it cannot generate non-convex portions of the Pareto front. Another limitation of VEGA is that its selection mechanism actually opposes the notion of Pareto optimality, because a solution that represents a good trade-off among all the objectives, but it is not the best in any of them won't be favored by its selection mech-

anism. Schaffer tried to overcome this issue by using some heuristic rules, such as *mating restrictions*, which imposes rules on the individuals that may recombine within the same subpopulation.

3.2 Non-Elitist Pareto-based Methods

Goldberg [14] proposed a sketch of the *non-dominated sorting* algorithm, which ranks the population of a GA using Pareto dominance. In this case, all non-dominated individuals should get the highest rank. Then, these solutions are discarded so that a new ranking is determined. The process continues until the whole population had been ranked. In order to maintain diversity, Goldberg suggested the use of what is now called a *density estimator*. This sort of approach blocks the selection bias of the GA, thus avoiding convergence to a single solution, and it is necessary so that a MOEA can generate different elements of the Pareto optimal set in a single run. Goldberg suggested to use *fitness sharing* [68] as a density estimator. This technique penalizes an individual's fitness by dividing it by a *niche count*, which estimates how crowded is the individual's neighborhood. A neighborhood is represented by a hypersphere of radius σ_{share} surrounding the current position of an individual. So, as more individuals lie within this radius (i.e., within the same niche), the lower the fitness of an individual. Evidently, σ_{share} is a parameter to which the performance of fitness sharing is highly sensitive, but some schemes to estimate it in multimodal optimization have been available since the late 1980s [69].

It turns out that non-dominated sorting is not the only scheme possible to rank solutions based on Pareto optimality. So, in this section we describe MOEAs that adopt a Pareto-based selection mechanism, but that do not incorporate elitism. These early Pareto-based MOEAs are easy to implement, but they are not very effective in problems having more than two or three objectives.

3.2.1 MOGA

The Multi-Objective Genetic Algorithm (MOGA) proposed by Fonseca and Fleming [70], ranks an individual according to the number of solutions that dominate it. This information is then used to compute its fitness, which is assumed to be maximized, ranging in the interval $[0, 1]$. Furthermore, the authors of this MOEA provide a polynomial equation to estimate the value of the niche radius, since MOGA uses fitness sharing as its density estimator. Parent selection is restricted to individuals with similar fitness values.

MOGA was a very popular MOEA in the mid-1990s, and was widely used in automatic control [71].

3.2.2 NSGA

The Non-dominated Sorting Genetic Algorithm (NSGA), proposed by Srinivas and Deb [72], closely follows Goldberg's original proposal. However, instead of ranking the population, a large dummy fitness value is assigned to the first layer of non-dominated solutions (i.e., those which are non-dominated with respect to the entire population). The first layer is then removed, and a second set of non-dominated solutions is identified. Such solutions receive dummy fitnes values which are lower than those from the first layer. This process is repeated until the entire population had been classified. Fitness values are shared among solutions lying in the same layer. Parents are chosen using a stochastic remainder proportionate selection based on the individuals' fitness values. Unlike MOGA, fitness sharing is applied in this case, in decision variable space.

NSGA has been criticized mainly for the need of specifying the parameter σ_{share} and for its high computational complexity of $O(k|P|^3)$, where $|P|$ is the population size and k is the number of objectives. The few comparative studies of MOEAs performed in the late 1990s indicated that NSGA was slower than MOGA and it produced solutions of inferior quality [73].

3.2.3 NPGA

The Niched-Pareto Genetic Algorithm (NPGA), introduced by Horn et al. [74], can be seen as a variation of the non-dominated sorting algorithm. In this approach, parent selection is performed through *Pareto domination tournaments*, in which two candidates are picked at random and compared against a (randomly selected) sample of the population. If one candidate is dominated by this sample, and the other is not, the latter is selected. Otherwise, the candidate with the smallest niche count in objective space becomes the winner. The main disadvantage of this MOEA is that it introduces a new parameter: the tournament size, which is critical to the performance of the MOEA. A tournament size corresponding to the 10% of the total population size is suggested by the authors of this MOEA.

NPGA is the fastest non-elitist Pareto-based MOEA. Nevertheless, it is not as effective as MOGA [73].

3.3 Elitist Pareto-based Methods

Within this group, we include the elitist methods that also implement Goldberg's Pareto ranking idea, but the algorithms in this case are more efficient and more effective than the non-elitist ones. The density estimators adopted in this group are also more sophisticated and, in some cases, even parameter-free. In fact, some of the MOEAs in this group are still in use today. Their main limitation is that most of them are not effective when

dealing with problems having more than 3 objectives (the so-called *many-objective optimization problems*). The main reason for this ineffectiveness is related to their density estimators which, in most cases, were designed for problems having only two objectives. Since Pareto ranking will quickly become ineffective in the presence of many objectives (this is because most of the population will quickly become non-dominated), the search in such problems is solely guided by the density estimator [75].

3.3.1 SPEA and SPEA2

The Strength Pareto Evolutionary Algorithm (SPEA), proposed by Zitzler and Thiele [76], mixes different mechanisms from previous MOEAs. SPEA incorporates elitism through the use of an external archive of the non-dominated solutions discovered so far. This archive is bounded to a maximum size using a clustering technique (the average linkage method). Parent selection is accomplished by binary tournaments, where the union of the main population and the external archive are considered. The fitness (or *strength*) of an individual in the external archive is calculated as the number of population members that it covers divided by the size of the population, plus one. The fitness of a member of the main population corresponds to the accumulated strengths of the external individuals that cover it, plus one. Here, the *cover* relation relies on Pareto dominance, and it is defined as follows: given $\mathbf{x}, \mathbf{y} \in \mathcal{X}$, it is said that $\mathbf{x}$ covers $\mathbf{y}$ iff $\mathbf{x} \prec \mathbf{y}$ or $\mathbf{x} = \mathbf{y}$. The aim of this fitness assignment scheme is to maintain diversity in the population. Ideally, the individuals of the external archive will cover the same number of population members. However, this only works if the external archive is uniformly distributed by the clustering method.

In 2001, an improved version of SPEA, called SPEA2, was proposed by Zitzler et al. [77], who reported the following drawbacks of the original version:

1. SPEA behaves like a random search algorithm when the population members have identical fitness values.
2. The search is stuck when there are many non-dominated solutions.
3. The spread is negatively affected because the extreme points are lost.
4. The archive's size may vary over time.

To tackle issue (1), the fitness assignment mechanism was changed and unified for both the main population and the external archive. A fitness value considers density information and both dominating and dominated solutions. Issues (2) and (3) are resolved in the archive truncation mechanism. The clustering technique is replaced by a density estimator based on the k^{th} nearest neighbor method, which does not lose boundary points.

Finally, to solve (4), if the number of non-dominated solutions from the population and the external archive is less than a limit size, then the dominated solutions with the best fitness values are copied to the external archive.

Comparative studies have indicated that SPEA2 is a very competitive MOEA, normally matching the performance of NSGA-II, but providing a better distribution of solutions, particularly when the number of objectives is higher than two.

3.3.2 NSGA-II

The Non-dominated Sorting Genetic Algorithm II (NSGA-II), proposed by Deb et al. [78], alleviates the main drawbacks of the original NSGA, although the new algorithm is really quite different. In this case, a more efficient non-dominated sorting algorithm is implemented and fitness sharing is replaced by a new density estimator called *crowding distance*, which operates in objective space and can be seen as the perimeter of the cuboid formed by the nearest neighbors surrounding a particular solution. The crowding distance is calculated as the average distance between the two points on either side of a solution along each objective. For comparison purposes, instead of fitness values, a preference relation is used, favoring those individuals with 1) lower ranks and 2) higher crowding distances. This preference relation is used in both selection mechanisms. Moreover, for the mating pool, stochastic remainder proportionate selection is replaced by binary tournaments. Finally, the survival selection is now elitist since the best half from the union of parents and offspring is retained.

The elegance, effectiveness and efficiency of NSGA-II made it a standard in evolutionary multi-objective optimization for more than ten years.

3.3.3 PAES

The Pareto Archived Evolution Strategy (PAES), proposed by Knowles and Corne in [79], is perhaps the most simple MOEA that one can possibly design. It consists of a $(1+1)$-ES (i.e., a single parent is mutated to produce an offspring). If the offspring dominates its parent, it is stored in an external archive and it becomes the parent in the next iteration. The most interesting aspect of this MOEA is its external archive, which adopts a density estimator called *adaptive grid*. In this approach, objective space is partitioned into l^k hypercubes, where l is a parameter that indicates the number of subdivisions of the space. The number of solutions in a hypercube is then used as an approximate form of a niche count.

The main problem of the adaptive grid is that it was conceived only for two objectives and its generalization to any number of objectives does not seem possible. Also, the fact that PAES does not adopt a crossover operator, limits its effectiveness in certain types of MOPs (e.g., when the Pareto front

is disconnected).

3.3.4 Micro-Genetic Algorithm

The Micro-Genetic Algorithm (micro-GA) for multiobjective optimization was proposed by Coello and Toscano [80]. This algorithm adopts three forms of elitism and a slight variant of the adaptive grid of PAES. Its main advantage is its efficiency. In their original experiments, its authors were able to show that the micro-GA was up to one order of magnitude faster than NSGA-II and it produced solutions of a similar quality. The main drawback of this approach was that it required a high number of parameters (eight, from which at least three played a fundamental role in its performance).

The Micro-Genetic Algorithm for Multi-Objective Optimization 2 (called μGA^2), introduced by Toscano and Coello [81], is an improved version of its ancestor. This is the only fully self-adaptive MOEA that has been proposed so far. The main motivation for the development of the μGA^2 was to eliminate the eight parameters required by the original algorithm. The μGA^2 uses on-line adaptation mechanisms that make unnecessary the fine-tuning of any of its parameters. Additionally, it can decide when to stop (no maximum number of generations has to be provided by the user). The only parameter that it requires is the size of the external archive (although there is a default value for this parameter).

3.4 Elitist Non-Pareto-based Methods

Within this group, we include the recent trends in the area, which involve MOEAs that are elitist and do not necessarily rely on Pareto dominance. These new approaches were motivated either by efficiency or by the need to tackle many-objective optimization problems.

3.4.1 Decomposition-based Methods

These methods transform a MOP into several single-objective subproblems, which are solved simultaneously, exploiting the population-based nature of MOEAs. Each subproblem is associated with a different target direction or weight vector. In order to obtain a wide variety of solutions, weight vectors should be uniformly distributed in the $[0, 1]^k$ space. Though, this does not necessarily imply that approximations to the Pareto optimal front will exhibit a uniform distribution. A popular method for generating weight vectors can be found in [82]. Moreover, decomposition is performed through an aggregating function, which maps an objective vector to a scalar. Some aggregating functions are compatible with a relaxed form of Pareto dominance, such as the Chebyshev function [62].

The most typical method in this category is the Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D), proposed by Zhang

and Li [83]. This approach is inspired on a mathematical programming technique called Normal Boundary Intersection (NBI) [82]. MOEA/D consists of a framework of aggregating functions in which variation operators are applied locally. An individual is associated with a weight vector/subproblem. Therefore, each member of the population keeps a neighborhood structure of a certain size. The vicinity is imposed by the closeness of the weight vectors. For a particular individual, two parents are randomly selected from its neighborhood. These parents generate a new offspring by using variation operators. The offspring is evaluated using the MOP and the aggregating function. The new solution replaces the current individual or its neighborhood if it minimizes their corresponding subproblems. In this way, the population is composed of the best solutions found so far for each subproblem.

MOEA/D is not only one of the most competitive MOEAs in current use but it is also very efficient (computationally speaking).

3.4.2 Indicator-based Methods

Performance indicators [84] evaluate the quality of an approximation to the Pareto optimal front, regarding convergence or diversity. Performance indicators have been mainly used to compare the effectiveness of optimizers, and recently, an important trend is their incorporation into the selection mechanism of MOEAs.

The hypervolume [85], also known as the Lebesgue measure or $\mathcal{S}$ metric, is one of the most popular performance indicators. The hypervolume of a set of solutions measures the size of the portion of objective space that is dominated by such solutions collectively and bounded by a reference point. A nice mathematical property of the hypervolume is that its maximization is equivalent to reaching the Pareto optimal set [86]. This has been experimentally validated [87, 88] and it has been observed that such solutions also have a good distribution along the Pareto optimal front.

Nevertheless, the main drawback of using the hypervolume is its high computational cost, which grows exponentially with the number of objectives. Moreover, when this performance indicator is coupled to a MOEA the selection of solutions is not straightforward. This indicator operates on a set of solutions, and the selection operator considers only one solution at a time. Therefore, when using the hypervolume to select solutions, a fitness assignment strategy is required. The most common is the *contribution to the hypervolume*: $\Delta_{\mathcal{S}}(\mathbf{a}, A) := \mathcal{S}(A) - \mathcal{S}(A \setminus \{\mathbf{a}\})$, where A is the set, $\mathbf{a}$ is a solution and $\mathcal{S}$ is the hypervolume indicator. In the following, we briefly illustrate its use within a MOEA that is representative of this group.

The $\mathcal{S}$ Metric Selection Evolutionary Multi-objective Algorithm (SMS-EMOA), proposed by Emmerich et al. [88], borrows ideas from NSGA-II and the archiving strategies proposed by Knowles, Corne and Fleischer [87]. Par-

ent selection is performed at random, whereas the survival selection mechanism relies on a (μ+1) scheme. The core idea of this algorithm is to integrate new solutions into the population if replacing a member of the population increases the hypervolume covered by the entire population. SMS-EMOA ranks the population according to NSGA-II, and the density estimator corresponds to the hypervolume contribution. At each iteration, the individual belonging to the worst rank and having the lowest hypervolume contribution is removed from the population. It is worth mentioning that the hypervolume contribution is calculated only when there is more than one solution having the worst rank. Additionally, an important feature of SMS-EMOA is that it is independent of the choice of the reference point and the scaling of the objectives. Also, it is guaranteed that the covered hypervolume of a population cannot decrease through iterations. SMS-EMOA is considered one of the most effective MOEAs for generating high-quality solutions.

3.4.3 Reference Point-based Methods

These methods guide the population towards a set of predefined reference points, ensuring diversity. Therefore, they are usually coupled with other strategies to achieve convergence, such as Pareto dominance. The reference points can either be supplied by the user or predefined in a structured manner by some method. A popular choice is the approach proposed by Das and Dennis in [82] (note that this is the same method for generating the weights in MOEA/D), in which the reference points are equally spaced on a normalized hyper-plane. There are several representative algorithms of this category (see for example [75]).

The most representative MOEA within this group is the Non-dominated Sorting Genetic Algorithm III (NSGA-III). This algorithm was proposed by Deb and Jain [89], and it is an extension of NSGA-II specifically designed to deal with many-objective optimization problems. NSGA-III still uses the non-dominated sorting algorithm for ranking the population. However, the crowding distance is replaced by a niching strategy that requires a set of reference points, which are adaptively updated according to the extent of the population. In NSGA-III, the population is normalized and associated with the lines passing through the origin and the reference points. Those individuals having the closest perpendicular distance to segregated lines are chosen for the next generation. Unlike NSGA-II, the parent selection is conducted by random sampling. Experimental results have shown that this approach can be used for solving problems having up to 15 objectives, outperforming MOEA/D.

3.5 Applications

Today, there exists a very important volume of applications of MOEAs in a wide variety of domains, including robotics [90], design and manufacture [91] scheduling [92], chemistry [93] physics [94] and medicine [95] among many others.

4 Conclusions

This chapter has provided the fundamentals on the use of evolutionary algorithms in optimization (both for single-objective and for multi-objective problems).

The chapter is divided in two parts. In the first part, a short description of the four most important evolutionary algorithms and their use in single-objective optimization is provided, together with their components and some of their applications. In the second part, a short introduction to multi-objective optimization and the most representative evolutionary algorithms developed for solving such problems is provided. The discussion includes a small taxonomy of approaches and some sample applications.

5 List of Further Reading

For more information on evolutionary optimization in general, see:

- Dan Simon, "Evolutionary Optimization Algorithms. Biologically Inspired and Population-Based Approaches to Computer Intelligence", Wiley, New Jersey, USA, 2013, ISBN 978-0-470-93741-9.
- Patrick Siarry and Zbigniew Michalewicz (Editors), *Advances in Metaheuristics for Hard Optimization*, Springer, Berlin, Germany, 2007, ISBN 978-3-540-72959-4.

For more information on multi-objective optimization using evolutionary algorithms, see:

- Carlos A. Coello Coello, Gary B. Lamont and David A. Van Veldhuizen, "Evolutionary Algorithms for Solving Multi-Objective Problems", Second Edition, Springer-Verlag, New York, USA, September 2007, ISBN 978-0-387-33254-3.
- Slim Bechikh, Rituparna Datta and Abhishek Gupta (Editors), "Recent Advances in Evolutionary Multi-objective Optimization", Springer International Publishing, Switzerland, 2017, ISBN 978-3-319-42977-9.

Acknowledgements

The first author gratefully acknowledges support from CONACyT project no. 221551.

Abbreviations

CMA-ES Covariance Matrix Adaptation Evolution Strategy

EA Evolutionary Algorithm

EP Evolutionary Programming

ES Evolution Streategies

GA Genetic Algorithm

MOEA Multi-objective Evolutionary Algorithm

MOEA/D Multi-Objective Evolutionary Algorithm based on Decomposition

MOGA Multi-Objective Genetic Algorithm

MOP Multi-objective Optimization Problem

NBI Normal Boundary Intersection

NPGA Niched-Pareto Genetic Algorithm

NSGA Non-dominated Sorting Genetic Algorithm

PAES Pareto Archived Evolution Strategy

SMS-EMOA $\mathcal{S}$ Metric Selection Evolutionary Multi-objective Algorithm

SPEA Strength Pareto Evolutionary Algorithm

VEGA Vector Evaluated Genetic Algorithm

References

[1] Carlos A. Coello Coello, Carlos Segura, and Gara Miranda. History and philosophy of evolutionary computation. In Plamen Parvanov Angelov, editor, *Handbook on Computational Intelligence*, volume 2, chapter 14, pages 509–545. World Scientific, Singapore, 2016. ISBN 978-981-4675-04-8.

[2] W. D. Cannon. *The Wisdom of the Body.* Norton and Company, New York, NY, 1932.

[3] John H. Holland. Outline for a logical theory of adaptive systems. *Journal of the Association for Computing Machinery*, 9:297–314, 1962.

[4] Hans-Paul Schwefel. Kybernetische evolution als strategie der experimentellen forschung inder strömungstechnik. Dipl.-Ing. thesis, 1965. (in German).

[5] L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence through Simulated Evolution.* John Wiley & Sons, New York, USA, 1966.

[6] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[7] Kenneth V. Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential Evolution. A Practical Approach to Global Optimization.* Springer, Berlin, 2005. ISBN 3-540-20950-6.

[8] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, September 2003.

[9] Jason Brownlee. *Clever Algorithms. Nature-Inspired Programming Recipes.* LuLu, 2011. ISBN 978-1-4467-8506-5.

[10] Dipankar Dasgupta, editor. *Artificial Immune Systems and Their Applications.* Springer-Verlag, Berlin, 1999.

[11] John R. Koza. *Genetic Programming. On the Programming of Computers by Means of Natural Selection.* The MIT Press, Cambridge, Massachusetts, 1992.

[12] Singiresu S. Rao. *Engineering Optimization.* John Wiley & Sons, third edition, 1996.

[13] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing.* Springer, New York, USA, second edition, 2015. ISBN 978-3-662-44873-1.

[14] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[15] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, New York, third edition, 1996.

[16] Gilbert Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[17] L. J. Eshelman and J. D. Schaffer. Real-coded Genetic Algorithms and Interval Schemata. In Darrell L. Whitley, editor, *Foundation of Genetic Algorithms 2*, pages 187–202, San Mateo, CA, 1993. Morgan Kaufmann.

[18] Kalyanmoy Deb and Ram Bhushan Agrawal. Simulated Binary Crossover for Continuous Search Space. *Complex Systems*, 9:115–148, 1995.

[19] James E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*, pages 14–21, Hillsdale, NJ, USA, 1987. L. Erlbaum Associates Inc.

[20] Günter Rudolph. Convergence Analysis of Canonical Genetic Algorithms. *IEEE Transactions on Neural Networks*, 5:96–101, January 1994.

[21] Kalyanmoy Deb, Ashish Anand, and Dhiraj Joshi. A Computationally Efficient Evolutionary Algorithm for Real-Parameter Optimization. *Evolutionary Computation*, 10(4):371–395, Winter 2002.

[22] Jon T. Richardson, Mark R. Palmer, Gunar E. Liepins, and Mike Hilliard. Some guidelines for genetic algorithms with penalty functions. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 191–197, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

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

[24] Efrén Mezura-Montes, editor. *Constraint-Handling in Evolutionary Optimization*. Springer, Berlin, Germany, 2009. ISBN 978-3-642-00618-0.

[25] Martin Serpell and James E. Smith. Self-adaptation of mutation operator and probability for permutation representations in genetic algorithms. *Evolutionary Computation.*, 18(3):491–514, September 2010.

[26] Manuel López-Ibá nez, Jeremie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

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

[28] Sedigheh Mahdavi, Shahryar Rahnamayan, and Mohammad Ebrahim Shiri. Multilevel framework for large-scale global optimization. *Soft Computing*, 21(14):4111–4140, July 2017.

[29] Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Per Kristian Lehre, Pietro S. Oliveto, Dirk Sudholt, and Andrew M. Sutton. *Emergence of Diversity and Its Benefits for Crossover in Genetic Algorithms*, pages 890–900. Springer International Publishing, Cham, 2016.

[30] Lin Wang, Athanasios Kolios, Takafumi Nishino, Pierre-Luc Delafin, and Theodore Bird. Structural optimisation of vertical-axis wind turbine composite blades based on finite element analysis and genetic algorithm. *Composite Structures*, 153:123–138, October 1 2016.

[31] Hwayeon Song, Jamshid Ghaboussi, and Tae-Hyun Kwon. Architectural design of apartment buildings using the Implicit Redundant Representation Genetic Algorithm. *Automation in Construction*, 72:166–173, December 2016.

[32] Rejane de B. Araujo and Antonio A.R. Coelho. Filtered predictive control design using multi-objective optimization based on genetic algorithm for handling offset in chemical processes. *Chemical Engineering Research & Design*, 117:265–273, January 2017.

[33] Cristina Renzi. A genetic algorithm-based integrated design environment for the preliminary design and optimization of aeronautical piston engine components. *International Journal of Advanced Manufacturing Technology*, 86(9-12):3365–3381, October 2016.

[34] C.A.F. Costa, M.A.C. Pacheco, O.P. Vilela Neto, and M. Cremona. Optimization of the Electrical Efficiency of Graded Multilayer Organic Light-Emitting Diodes Supported by Genetic Algorithm. *Journal of Computational and Theoretical Nanoscience*, 11(6):1505–1511, June 2014.

[35] I. Rechenberg. *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog, 1973.

[36] H. P. Schwefel. *Evolution and Optimum Seeking*. Wiley Interscience, New York, 1995.

[37] Oliver Kramer. A Review of Constraint-Handling Techniques for Evolution Strategies. *Applied Computational Intelligence and Soft Computing*, 2010(1):1–11, January 2010. Article ID 185063.

[38] Efrén Mezura-Montes and Carlos A. Coello Coello. Constraint-Handling in Nature-Inspired Numerical Optimization: Past, Present and Future. *Swarm and Evolutionary Computation*, 1(4):173–194, December 2011.

[39] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 312–317, May 1996.

[40] Leidy Y. Serna, Miguel Angel Mananas, Jesus Marin, Alher Mauricio Hernandez, and Salvador Benito. Optimization techniques in respiratory control system models. *Applied Soft Computing*, 48:432–443, November 2016.

[41] Javier Barrachina, Piedad Garrido, Manuel Fogue, Francisco J. Martinez, Juan-Carlos Cano, Carlos T. Calafate, and Pietro Manzoni. Reducing emergency services arrival time by using vehicular communications and Evolution Strategies. *Expert Systems with Applications*, 41(4):1206–1217, March 2014.

[42] Anita Emmerstorfer-Augustin, Sandra Moser, and Harald Pichler. Screening for improved isoprenoid biosynthesis in microorganisms. *Journal of Biotechnology*, 235:112–120, October 10 2016.

[43] Georgios S. Papavasileiou and Dimos C. Charmpis. Seismic design optimization of multi-storey steel-concrete composite buildings. *Computers & Structures*, 170:49–61, July 1 2016.

[44] Jong-Hwan Kim and Hyun Myung. Evolutionary programming techniques for constrained optimization problems. *Transactions on Evolutionary Computation*, 1(2):129–140, July 1997.

[45] Libin Hong, John H. Drake, John R. Woodward, and Ender Özcan. Automatically designing more general mutation operators of evolutionary programming for groups of function classes using a hyper-heuristic. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 725–732, New York, NY, USA, 2016. ACM.

[46] P. Nagarathna and R. Manjula. Maximization of WSN Life Using Hybrid Evolutionary Programming. *International Journal of Wireless Information Networks*, 23(3):246–256, September 2016.

[47] Apolinar Velarde Martinez, Juan Antonio Nungaray Ornelas, Eunice Ponce de Leon Senti, and Juan Alejandro Montanez de la Torre. Planning and Allocation of Tasks in a Multiprocessor System as a Multi-Objective Problem and its Resolution Using Evolutionary Programming. *International Journal of Advanced Computer Science and Applications*, 7(3):349–360, March 2016.

[48] Alon Oz, Shany Hershkovitz, Nataly Belman, Ervin Tal-Gutelmacher, and Yoed Tsur. Analysis of impedance spectroscopy of aqueous supercapacitors by evolutionary programming: Finding DFRT from complex capacitance. *Solid State Ionics*, 288:311–314, May 2016.

[49] Vitaliy Feoktistov and Stefan Janaqi. Generalization of the Strategies in Differential Evolution. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004), 2004, Santa Fe, New Mexico, USA*, page 165a, New Mexico, USA, April 2004. IEEE Computer Society.

[50] Efrén Mezura-Montes, Jesús Velázquez-Reyes, and Carlos A. Coello Coello. Comparing Differential Evolution Models for Global Optimization. In Maarten Keijzer et al., editor, *2006 Genetic and Evolutionary Computation Conference (GECCO'2006)*, volume 1, pages 485–492, Seattle, Washington, USA, July 2006. ACM Press.

[51] Kenneth V. Price. An Introduction to Differential Evolution. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 79–108. McGraw-Hill, London, UK, 1999.

[52] Noha M. Hamza, Ruhul A. Sarker, and Daryl L. Essam. Differential Evolution with a mix of Constraint Consensus Methods for Solving a Real-World Optimization Problem. In *2012 IEEE Congress on Evolutionary Computation (CEC'2012)*, pages 2791–2797, Brisbane, Australia, June 10-15 2012. IEEE Press.

[53] Petr Bujok and Josef Tvrdík. *Adaptive Differential Evolution: SHADE with Competing Crossover Strategies*, pages 329–339. Springer International Publishing, Cham, 2015.

[54] Felipe Campelo and Moisés Botelho. Experimental investigation of recombination operators for differential evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 221–228, New York, NY, USA, 2016. ACM.

[55] Emerson Hochsteiner de Vasconcelos Segundo, Anderson Levati Amoroso, Viviana Cocco Mariani, and Leandro dos Santos Coelho. Economic optimization design for shell-and-tube heat exchangers by

a Tsallis differential evolution. *Applied Thermal Engineering*, 111:143–151, January 25 2017.

[56] Caglayan Balkaya, Yunus Levent Ekinci, Gokhan Gokturkler, and Secil Turan. 3D non-linear inversion of magnetic anomalies caused by prismatic bodies using differential evolution algorithm. *Journal of Applied Geophysics*, 136:372–386, January 2017.

[57] Urog Mlakar, Bozidar Potocnik, and Janez Brest. A hybrid differential evolution for optimal multilevel image thresholding. *Expert Systems with Applications*, 65:221–232, December 15 2016.

[58] A.E. Eiben and S.K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19 – 31, 2011.

[59] A. E. Eiben and S. K. Smit. *Evolutionary Algorithm Parameters and Methods to Tune Them*, pages 15–36. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[60] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, Jul 1999.

[61] K. L. Mills, J. J. Filliben, and A. L. Haines. Determining relative importance and effective settings for genetic algorithm control parameters. *Evolutionary Computation*, 23(2):309–342, June 2015.

[62] Kaisa M. Miettinen. *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, Boston, Massachusetts, 1999.

[63] Günter Rudolph and Alexandru Agapie. Convergence Properties of Some Multi-Objective Evolutionary Algorithms. In *Proceedings of the 2000 Conference on Evolutionary Computation*, volume 2, pages 1010–1016, Piscataway, New Jersey, July 2000. IEEE Press.

[64] M. P. Fourman. Compaction of Symbolic Layout using Genetic Algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 141–153. Lawrence Erlbaum, 1985.

[65] Neil H. Eklund and Mark J. Embrechts. Determining the Color-Efficiency Pareto Optimal Surface for Filtered Light Sources. In Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, and David Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 603–611. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

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

[67] J. David Schaffer. Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 93–100. Lawrence Erlbaum, 1985.

[68] David E. Goldberg and Jon Richardson. Genetic algorithm with sharing for multimodal function optimization. In John J. Grefenstette, editor, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49, Hillsdale, New Jersey, 1987. Lawrence Erlbaum.

[69] Kalyanmoy Deb and David E. Goldberg. An Investigation of Niche and Species Formation in Genetic Function Optimization. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 42–50, San Mateo, California, June 1989. George Mason University, Morgan Kaufmann Publishers.

[70] Carlos M. Fonseca and Peter J. Fleming. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423, San Mateo, California, 1993. University of Illinois at Urbana-Champaign, Morgan Kauffman Publishers.

[71] C.M. Fonseca and P.J. Fleming. Multiobjective genetic algorithms. In A.M.S. Zalzala and P.J. Fleming, editors, *Genetic Algorithms in Engineering Systems*, chapter 3, pages 63–78. The Institution of Electrical Engineers. Control Engineering Series 55, Bath, UK, 1997.

[72] N. Srinivas and Kalyanmoy Deb. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3):221–248, Fall 1994.

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

[74] Jeffrey Horn, Nicholas Nafpliotis, and David E. Goldberg. A Niched Pareto Genetic Algorithm for Multiobjective Optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, volume 1, pages 82–87, Piscataway, New Jersey, June 1994. IEEE Service Center.

[75] Bingdong Li, Jinlong Li, Ke Tang, and Xin Yao. Many-Objective Evolutionary Algorithms: A Survey. *ACM Computing Surveys*, 48(1), September 2015.

[76] Eckart Zitzler and Lothar Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, November 1999.

[77] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. In K. Giannakoglou, D. Tsahalis, J. Periaux, P. Papailou, and T. Fogarty, editors, *EUROGEN 2001. Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*, pages 95–100, Athens, Greece, 2001.

[78] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.

[79] Joshua D. Knowles and David W. Corne. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evol. Comput.*, 8(2):149–172, jun 2000.

[80] Carlos A. Coello Coello and Gregorio Toscano Pulido. Multiobjective Optimization using a Micro-Genetic Algorithm. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 274–282, San Francisco, California, 2001. Morgan Kaufmann Publishers.

[81] Gregorio Toscano Pulido and Carlos A. Coello Coello. The Micro Genetic Algorithm 2: Towards Online Adaptation in Evolutionary Multiobjective Optimization. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, pages 252–266, Faro, Portugal, April 2003. Springer. Lecture Notes in Computer Science. Volume 2632.

[82] Indraneel Das and J. E. Dennis. Normal-Boundary Intersection: A New Method for Generating the Pareto Surface in Nonlinear Multicriteria Optimization Problems. *SIAM Journal on Optimization*, 8(3):631–657, 1998.

[83] Qingfu Zhang and Hui Li. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, Dec 2007.

[84] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, April 2003.

[85] Eckart Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, November 1999.

[86] M. Fleischer. The Measure of Pareto Optima. Applications to Multiobjective Metaheuristics. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, pages 519–533, Faro, Portugal, April 2003. Springer. Lecture Notes in Computer Science. Volume 2632.

[87] Joshua D. Knowles, David W. Corne, and Mark Fleischer. Bounded Archiving using the Lebesgue Measure. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC'2003)*, volume 4, pages 2490–2497, Canberra, Australia, December 2003. IEEE Press.

[88] Michael Emmerich, Nicola Beume, and Boris Naujoks. An EMO Algorithm Using the Hypervolume Measure as Selection Criterion. In Carlos A. Coello Coello, Arturo Hernández Aguirre, and Eckart Zitzler, editors, *Evolutionary Multi-Criterion Optimization. Third International Conference, EMO 2005*, pages 62–76, Guanajuato, México, March 2005. Springer. Lecture Notes in Computer Science Vol. 3410.

[89] K. Deb and H. Jain. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, Aug 2014.

[90] Dan Zhang and Zhen Gao. Forward kinematics, performance analysis, and multi-objective optimization of a bio-inspired parallel manipulator. *Robotics and Computer-Intregrated Manufacturing*, 28(4):484–492, August 2012.

[91] C. Senthilkumar, G. Ganesan, and R. Karthikeyan. Parametric optimization of electrochemical machining of Al/15% SiC(p) composites using NSGA-II. *Transactions of Nonferrous Metals Society of China*, 21(10):2294–2300, October 2011.

[92] Arnaud Liefooghe, Matthieu Basseur, Jeremie Humeau, Laetitia Jourdan, and El-Ghazali Talbi. On optimizing a bi-objective flowshop scheduling problem in an uncertain environment. *Computers & Mathematics with Applications*, 64(12):3747–3762, December 2012.

[93] Clare Levene, Elon Correa, Ewan W. Blanch, and Royston Goodacre. Enhancing surface enhanced raman scattering (sers) detection of propranolol with multiobjective evolutionary optimization. *Analytical Chemistry*, 84(18):7899–7905, September 18 2012.

[94] Vicent Romero-Garcia, Juan Vicente Sanchez-Perez, and Luis Miguel Garcia-Raffi. Molding the Acoustic Attenuation in Quasi-Ordered Structures: Experimental Realization. *Applied Physics Express*, 5(8), August 2012. Article number 087301.

[95] Fernando Jimenez, Gracia Sanchez, and Jose M. Juarez. Multi-Objective Evolutionary Algorithms for Fuzzy Classification in Survival Prediction. *Artificial Intelligence in Medicine*, 60(3):197–219, March 2014.