An Introduction to Evolutionary Algorithms with Applications in Biometrics

Carlos A. Coello Coello
CINVESTAV-IPN
Evolutionary Computation Group
Dpto. de Ing. Elect./Secc. Computación
Av. IPN No. 2508, Col. San Pedro Zacatenco
México, D.F. 07300, MEXICO
ccoello@cs.cinvestav.mx

Abstract

This paper provides a brief introduction to evolutionary algorithms and to some of their representative applications in biometrics. Our discussion includes short descriptions of genetic algorithms, evolution strategies, evolutionary programming and genetic programming as well as some of the terminology commonly adopted in this area. Then, a few case studies involving applications of evolutionary algorithms in biometrics are analyzed. In the final part of the paper, some possible research directions are provided.

1 Introduction

The idea of using techniques based on the emulation of the mechanism of natural selection to solve problems can be traced as long back as the 1930s [12]. However, it was until the 1960s when the three main techniques based on this notion were developed. These approaches, which are now collectively denominated "evolutionary algorithms", have been very effective for single-objective optimization [34, 62, 29].

Biometrics is a discipline that measures and statistically analyses biological data. Recently, and in the context of information technology, the term has been adopted to refer to the technologies for measuring and analyzing human body characteristics such as fingerprints, eye retinas and irises, voice patterns, facial patterns and hand measurements, especially for authentication purposes [74]. Biometric applications involve several complex problems. For example, many current biometric applications are closely related to pattern recognition and image analysis [63]. The complexity of these problems (which tend to be approached using statistical techniques) makes attractive the use of heuristics such as evolutionary algorithms, which have been found to be very powerful in a wide variety of optimization and classification tasks [29, 34, 4].

The remainder of this paper is organized as follows. Section 2 provides some basic concepts related to evolutionary algorithms. In Section 3 attempts to summarize the material from the previous section, by providing a more general framework for studying evolutionary algorithms and discussing some of their main advantages. Section 4 discusses a few representative case studies of applications of evolutionary algorithms in biometrics. After that, we provide some possible future research directions in Section 5 and our conclusions in Section 6.

2 Basic Notions of Evolutionary Algorithms

The famous naturalist Charles Darwin defined *Natural Selection* or *Survival of the Fittest* as the *preservation of favorable individual differences and variations, and the destruction of those that are injurious* [18]. In nature, individuals have to adapt to their environment in order to survive in a process called *evolution*, in which those features that make an individual more suited to compete are preserved when it reproduces, and those features that make it weaker are eliminated. Such features are controlled by units called *genes* which form sets called *chromosomes*. Over subsequent generations not only the fittest individuals survive, but also their fittest genes which are transmitted to their descendants during the sexual recombination process which is called *crossover*.

Early analogies between the mechanism of natural selection and a learning (or optimization) process led to the development of the so-called "evolutionary algorithms" (EAs) [3], in which the main goal is to simulate the evolutionary process in a computer. There are three main paradigms within evolutionary algorithms, whose motivations and origins were independent from each other: evolution strategies [62], evolutionary programming [32], and genetic al-

gorithms [40]. Additionally, some authors consider genetic programming [47] as another paradigm, although this can also be seen as a special type of genetic algorithm. Each of these four types of evolutionary algorithm will be discussed next in more detail.

2.1 Evolution Strategies

When working towards his PhD degree in engineering at the Technical University of Berlin, Ingo Rechenberg came across some optimization problems in hydrodynamics that could not be solved using traditional mathematical programming techniques [58]. This led him to the development of a very simple optimization algorithm which consisted of applying a set of random changes to a reference solution. The approach was later called "evolution strategy" and it was formally introduced in 1964 [30]. The original evolution strategy was called $(1+1)$-ES, because it consisted of a single parent that was mutated (i.e., subject to a random change) to produce an offspring. Then, the parent was compared to its offspring and the best from them was selected to become parent for the following iteration (or generation).

In the original (1+1)-EE, a new individual was produced using:

$$\bar{x}^{t+1} = \bar{x}^t + N(0, \bar{\sigma})$$

where *t* refers to the current *generation* (or iteration) and $N(0, \bar{\sigma})$ is a vector of independent Gaussian numbers with median zero and standard deviation $\bar{\sigma}$. It is important to emphasize that an "individual" in an evolution strategy contains the set of decision variables of the problem. No encoding is used in this case. So, if the decision variables are real numbers, such real numbers are directly put together as a single vector for each individual.

Let's consider the following example of a (1+1)-ES:

Let us assume that we want to **maximize**:

$$f(x_1, x_2) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$$

where: $-2.048 \leq x_1, x_2 \leq 2.048$

Now, let us suppose that our population consists of the following (randomly generated) individual:

$$(\bar{x}^t, \bar{\sigma}) = (-1.0, 1.0), (1.0, 1.0)$$

Let us now suppose that the mutations generated are the following:

$$x_1^{t+1} = x_1^t + N(0, 1.0) = -1.0 + 0.61 = -0.39$$
$$x_2^{t+1} = x_2^t + N(0, 1.0) = 1 + 0.57 = 1.57$$

Now, we compare the parent with its offspring:

Parent: $f(x_t) = f(-1.0, 1.0) = 4.0$
Child: $f(x_{t+1}) = f(-0.39, 1.57) = 201.416$

Since: $201.416 > 4.0$
the offspring will replace its parent in the following generation.

Rechenberg [59] stated a rule for adjusting the standard deviation in a deterministic way such that the evolution strategy could converge to the global optimum. This is now known as the "1/5 success rule", and it consists of the following:

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

where n is the number of decision variables, t is the current generation, p_s is the relative frequency of successful mutations (i.e., those mutations in which the offspring replaced its parent) measured over a certain period of time (e.g., $10n$ individuals) and $c = 0.817$ (this value was theoretically derived by Schwefel [62]). $\sigma(t)$ is adjusted at every n mutations.

Over the years, several other variations of the original evolution strategy were proposed, after the concept of population (i.e., a set of solutions) was introduced. The most recent versions of the evolution strategy are the $(\mu + \lambda)$-ES and the (μ, λ)-ES. In both cases, μ parents are mutated to produce λ offspring. However, in the first case (+ selection), the μ best individuals are selected from the union of parents and offspring. In the second case (i.e., , selection), the best individuals are selected only from the offspring produced.

In modern evolution strategies, not only the decision variables of the problem are evolved, but also the parameters of the algorithm itself (i.e., the standard deviations). This is called "self-adaptation". Parents are mutated using:

$$\sigma'(i) = \sigma(i) \times exp(\tau' N(0,1) + \tau N_i(0,1))$$

$$x'(i) = x(i) + N(0, \sigma'(i))$$

where τ and τ' are proportionality constants that are defined in terms of n.

Also, modern evolution strategies allow the use of recombination (either sexual, when only 2 parents are involved, or panmictic, when more than 2 parents are involved in the generation of the offspring).

Some representative applications of evolution strategies are the following [62]:

- Routing and networking.

- Biochemistry.
- Optics.
- Engineering design.
- Magnetism.

2.2 Evolutionary Programming

Lawrence J. Fogel introduced in the 1960s an approach called "evolutionary programming", in which intelligence is seen as an adaptive behavior [31, 32].

Evolutionary programming emphasizes the behavioral links between parents and offspring, instead of trying to emulate some specific genetic operators (as in the case of the genetic algorithm [34]).

The basic algorithm of evolutionary programming is very similar to that of the evolution strategy. A population of individuals is mutated to generate a set of offspring. However, in this case, there are normally several types of mutation operators and no recombination (of any type), since evolution is modelled at the species level in this case and different species do not interbreed. Another difference with respect to evolution strategies is that in this case, each parent produces exactly one offspring. Also, the decision of whether or not a parent will participate in the selection process is now determined in a probabilistic way, whereas in the evolution strategy this is a deterministic process. Finally, no encoding is used in this case (similarly to the evolution strategy) and emphasis is placed on the selection of the most appropriate representation of the decision variables.

We will now show an example of the way in which evolutionary programming works. Let us consider the finite automaton from Figure 1. The transition table corresponding to this automaton is the following:

Current State	**A**	**A**	**B**	**B**	**C**	**C**
Input Symbol	0	1	1	0	0	1
Next State	B	c	B	C	C	A
Output Symbol	a	b	c	b	a	b

Considering the type of problem at hand, several mutation operators are possible. For example: change an output symbol, change a transition, add a state, delete a state and change the initial state. The goal is to make this automaton able to recognize a certain set of inputs (i.e., a certain regular expression) without making a single mistake.

Some representative applications of evolutionary programming are the following [29]:

- Forecasting.
- Generalization.
- Games.
- Automatic control.
- Traveling salesperson problem.
- Route planning.
- Pattern recognition.
- Neural networks training.

2.3 Genetic Algorithms

Genetic algorithms (originally denominated "genetic reproductive plans") were introduced by John H. Holland in the early 1960s [38, 39]. The main motivation of this work was the solution of machine learning problems.

Genetic algorithms emphasize the importance of sexual recombination (which is the main operator) over the mutation operator (which is used as a secondary operator). They also use probabilistic selection (like evolutionary programming and unlike evolution strategies).

Some of the basic terminology used by the genetic algorithms (GAs) community is the following [37]:

- A **chromosome** is a data structure that holds a "string" of task parameters, or genes. This string may be stored, for example, as a binary bit-string (binary representation) or as an array of integers (floating point o real-coded representation) that represent a floating point number. This chromosome is analogous to the base-4 chromosomes present in our own DNA. Normally, in the GA community, the haploid model of a cell is assumed (one-chromosome individuals). However, diploids have also been used in the specialized literature [34].
- A **gene** is a subsection of a chromosome that usually encodes the value of a single parameter (i.e., a decision variable).
- An **allele** is the value of a gene. For example, for a binary representation each gene may have an allele of 0 or 1, and for a floating point representation, each gene may have an allele from 0 to 9.
- A **schema** (plural **schemata**) is a pattern of gene values in a chromosome, which may include "do not care" states (represented by a # symbol). Thus, in a binary chromosome, each schema can be specified by a string of the same length as the chromosome, with each character being one of $\{ 0, 1, \# \}$. A particular chromosome is said to "contain" a particular schema if it matches the scheme (e.g. chromosome 01101 matches schema #1#0#).

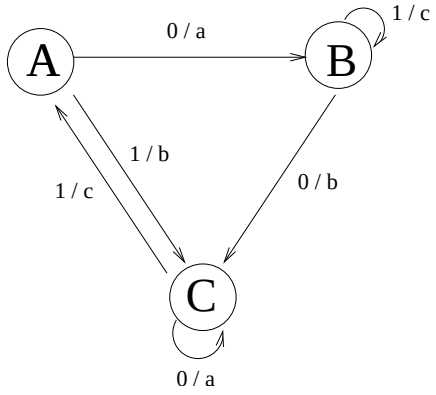

Figure 1. Finite automaton with 3 states. Symbols to the left of "/" are input symbols. Symbols to the right of "/" are output symbols. The initial state is A.

- If the solution of a problem can be represented by a set of N real-valued parameters, then the job of finding this solution can be thought of as a search in an N-dimensional space. This region is simply referred to as the **search space** of the problem.

- The **fitness** of an individual is a value that reflects its performance (i.e., how well solves a certain task). A **fitness function** is a mapping of the chromosomes in a population to their corresponding fitness values. A **fitness landscape** is the hypersurface obtained by applying the fitness function to every point in the search space.

- A **building block** is a small, tightly clustered group of genes which have co-evolved in such a way that their introduction into any chromosome will be likely to give increased fitness to that chromosome. The **building block hypothesis** [34] states that GAs generate their solutions by first finding as many building blocks as possible, and then combining them together to give the highest fitness.

- **Deception** is a condition under which the combination of good building blocks leads to reduced fitness, rather than increased fitness. This condition was proposed by Goldberg [34] as a reason for the failure of GAs on certain tasks.

- **Elitism** (or an elitist strategy) is a mechanism which ensures that the chromosomes of the highly fit member(s) of the population are passed on to the next generation without being altered by any genetic operator. The use of elitism guarantees that the maximum fitness of the population never decreases from one generation to the next, and it normally produces a faster convergence of the population. More important yet is the fact that it has been (mathematically) proven that elitism is necessary in order to be able to guarantee convergence of a simple genetic algorithm towards the global optimum [61].

- **Epistasis** is the interaction between different genes in a chromosome. It is the extent to which the contribution to fitness of one gene depends on the values of other genes. Geneticists use this term to refer to a "masking" or "switching" effect among genes, and a gene is considered to be "epistatic" if its presence suppresses the effect of a gene at another locus. This concept is closely related to deception, since a problem with high degree of epistasis is deceptive, because building blocks can not be formed. On the other hand, problems with little or no epistasis are trivial to solve (hill climbing is sufficient).

- **Exploitation** is the process of using information gathered from previously visited points in the search space to determine which places might be profitable to visit next. Hill climbing is an example of exploitation, because it investigates adjacent points in the search space, and moves in the direction giving the greatest increase in fitness. Exploitation techniques are good at finding local minima (or maxima). The GA uses crossover as an exploitation mechanism.

- **Exploration** is the process of visiting entirely new regions of a search space, to see if anything promising may be found there. Unlike exploitation, exploration involves leaps into unknown regions. Random search is an example of exploration. Problems which have many local minima (or maxima) can sometimes only be solved using exploration techniques such as random search. The GA uses mutation as an exploration mechanism.

- A **genotype** represents a potential solution to a problem, and is basically the string of values chosen by the user, also called chromosome.

- A **phenotype** is the meaning of a particular chromosome, defined externally by the user.

- Genetic drift is the name given to the changes in gene/allele frequencies in a population over many generations, resulting from chance rather than from selection. It occurs most rapidly in small populations and can lead to some alleles to become extinct, thus reducing the genetic variability in the population.

- A **niche** is a group of individuals which have similar fitness. Normally in multiobjective and multimodal optimization, a technique called **sharing** is used to reduce the fitness of those individuals who are in the same niche, in order to prevent the population to converge to a single solution, so that stable subpopulations can be formed, each one corresponding to a different objective or peak (in a multimodal optimization problem) of the function [21].

The basic operation of a Genetic Algorithm is illustrated in the following segment of pseudo-code [11]:

```
generate initial population, G(0);
evaluate G(0);
t:=0;
repeat
   t:=t+1;
   generate G(t) using G(t-1);
   evaluate G(t);
until a solution is found
```

First, an initial population is randomly generated. The individuals of this population will be a set of chromosomes or strings of characters (letters and/or numbers) that represent all the possible solutions to the problem.

One aspect that has great importance in the case of the genetic algorithm is the encoding of solutions. Traditionally, a binary encoding has been adopted, regardless of the type of decision variables of the problem to be solved [34]. Holland [40] provides some theoretical and biological arguments for using a binary encoding. However, over the years, other types of encodings have been proposed, including the use of vectors of real numbers and permutations, which lend themselves as more "natural" encodings for certain types of optimization problems [49, 60].

Once an appropriate encoding has been chosen, we apply a *fitness function* to each one of these chromosomes in order to measure the quality of the solution encoded by the chromosome. Knowing each chromosome's fitness, a *selection* process takes place to choose the individuals (presumably, the fittest) that will be the parents of the following generation. The most commonly used selection schemes are the following [35]:

- *Proportionate Reproduction:* This term is used generically to describe several selection schemes that choose individuals for birth according to their objective function values f. In these schemes, the probability of selection p of an individual from the ith class in the tth generation is calculated as

$$p_{i,t} = \frac{f_i}{\sum_{j=1}^{k} m_{j,t} f_j} \tag{1}$$

  where k classes exist and the total number of individuals sums to n. Several methods have been suggested for sampling this probability distribution, including Monte Carlo or *roulette wheel* selection [42], *stochastic remainder* selection [9, 10], and *stochastic universal* selection [6, 36].

- *Ranking Selection:* In this scheme, proposed by Baker [5] the population is sorted from best to worst, and each individual is copied as many times as it can, according to a non-increasing assignment function, and then proportionate selection is performed according to that assignment.

- *Tournament Selection:* The population is shuffled and then is divided into groups of k elements from which the best individual (i.e., the fittest) will be chosen. This process has to be repeated k times because on each iteration only m parents are selected, where

$$m = \frac{population\ size}{k}$$

  For example, if we use binary tournament selection ($k = 2$), then we have to shuffle the population twice, since in each stage half of the parents required will be selected. The interesting property of this selection scheme is that we can guarantee multiple copies of the fittest individual among the parents of the next generation.

- **Steady State Selection:** This is the technique used in Genitor [72], which works individual by individual, choosing an offspring for birth according to linear ranking, and choosing the currently worst individual for replacement. In steady-state selection only a few individuals are replaced in each generation: usually a small number of the least fit individuals are replaced by offspring resulting from crossover and mutation of the fittest individuals. This selection scheme is normally used in evolving rule-based systems in which

1	0	0	1	1	0	1

Figure 2. Example of the binary encoding traditionally adopted with the genetic algorithm.

incremental learning (and remembering what has already been learned) is important and in which members of the population collectively (rather than individually) solve the problem at hand [51].

After being selected, *crossover* takes place. During this stage, the genetic material of a pair of individuals is exchanged in order to create the population of the next generation. There are three main ways of performing crossover:

1. *Single-point crossover*: A position of the chromosome is randomly selected as the crossover point as indicated in Figure 3.

2. *Two-point crossover*: Two positions of the chromosome are randomly selected as to exchange chromosomic material, as indicated in Figure 4.

3. *Uniform crossover*: This is a relatively recent crossover operator proposed by Syswerda [66] which can be seen as a generalization of the two previous crossover techniques explained in this paper. In this case, for each bit in the first offspring it decides (with some probability p) which parent will contribute its value in that position. The second offspring would receive the bit from the other parent. See an example of 0.5-uniform crossover in Figure 5. Although for some problems uniform crossover presents several advantages over other crossover techniques [66], in general, one-point crossover seems to be a bad choice, but there is no clear winner between two-point and uniform crossover [48].

Mutation is another important genetic operator that randomly changes a gene of a chromosome. If we use a binary representation, a mutation changes a 0 to 1 and viceversa. An example of how mutation works is displayed in Figure 6. This operator allows the introduction of new chromosomic material to the population and, from the theoretical perspective, it assures that—given any population—the entire search space is connected [11].

If we knew in advance the final solution, it would be trivial to determine how to stop a genetic algorithm. However, as this is not normally the case, we have to use one of the two following criteria to stop the GA: either give a fixed number of generations in advance, or verify when the population has become homogeneous (i.e., all or most of the individuals have the same fitness).

Traditionally, genetic algorithms do not have a self-adaptation mechanism. Therefore, one of their main drawbacks is that their parameters tend to be fine-tuned in an empirical manner.

Some representative applications of genetic algorithms are the following [34]:

- Optimization (numerical, combinatorial, etc.).
- Machine learning.
- Databases (optimization of queries, etc.).
- Pattern recognition.
- Grammar generation.
- Robot motion planning.
- Forecasting.

2.4 Genetic Programming

One of the original goals of artificial intelligence (AI) was the automatic generation of computer programs that could produce a desired task given a certain input. During several years, such a goal seemed too ambitious since the size of the search space increases exponentially as we extend the domain of a certain program and, consequently, any technique will tend to produce programs that are either invalid or highly inefficient.

Some early evolutionary algorithms were attempted in automatic programming tasks, but they were unsuccessful and were severly criticized by some AI researchers [29]. Over the years, researchers realized that the key issue for using evolutionary algorithms in automatic programming tasks was the encoding adopted. In this regard, Koza [47] suggested the use of a genetic algorithm with a tree-based encoding. In order to simplify the implementation of such an approach, the original implementation of this sort of approach (which was called "genetic programming") was done under LISP, taking advantage of the fact that such programming language has a built-in parser.

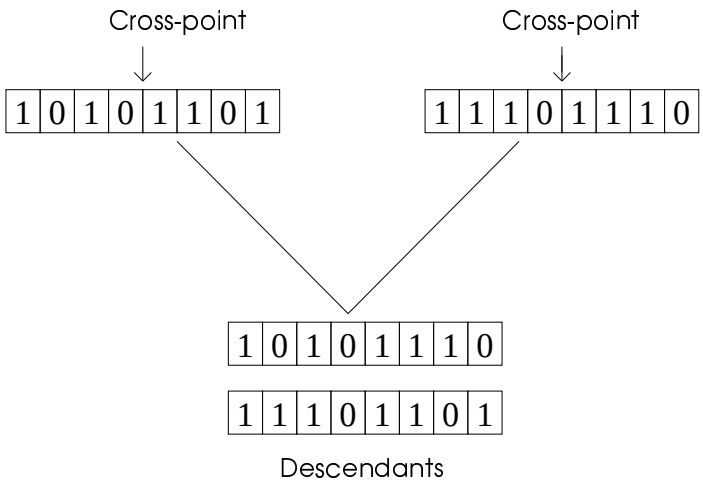

Figure 3. Use of a single-point crossover between two chromosomes. Notice that each pair of chromosomes produces two descendants for the next generation. The cross-point may be located at the string boundaries, in which case the crossover has no effect and the parents remain intact for the next generation.

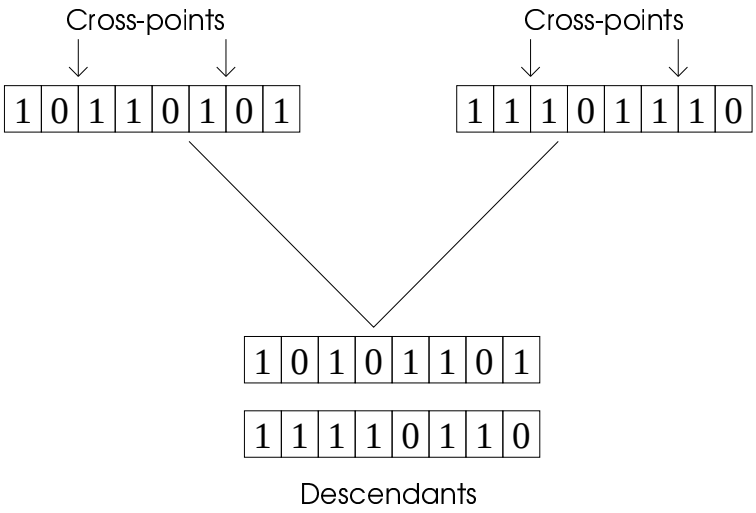

Figure 4. Use of a two-point crossover between two chromosomes. In this case the genes at the extremes are kept, and those in the middle part are exchanged. If one of the two cross-points happens to be at the string boundaries, a single-point crossover will be performed, and if both are at the string boundaries, the parents remain intact for the next generation.

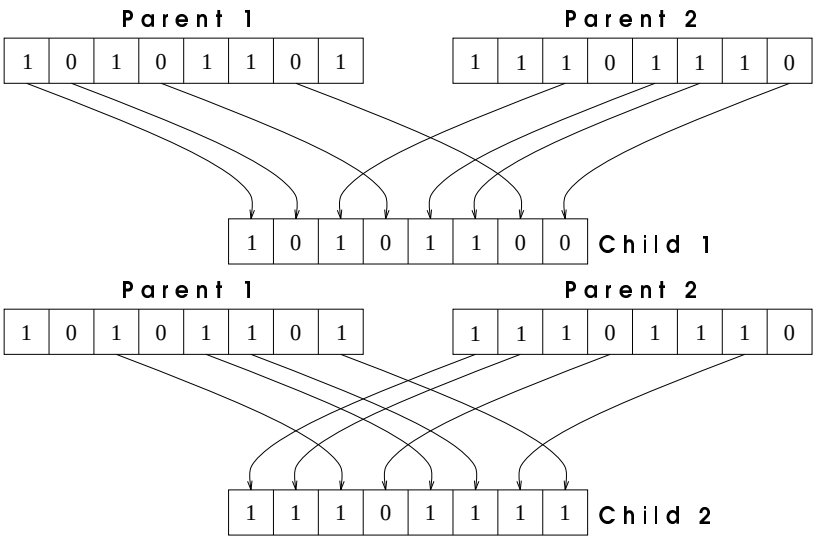

Figure 5. Use of 0.5-uniform crossover (using 50% probability) between two chromosomes. Notice how half of the genes of each parent goes to each of the two children. First, the bits to be copied from each parent are selected randomly using the probability desired, and after the first child is generated, the same values are used to generate the second child, but inverting the source of procedence of the genes.

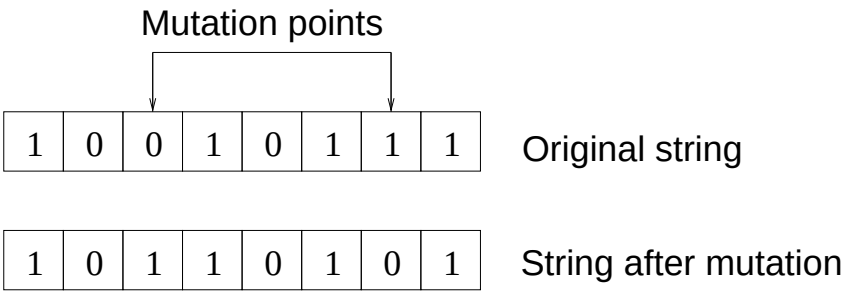

Figure 6. An Example of mutation using binary representation.

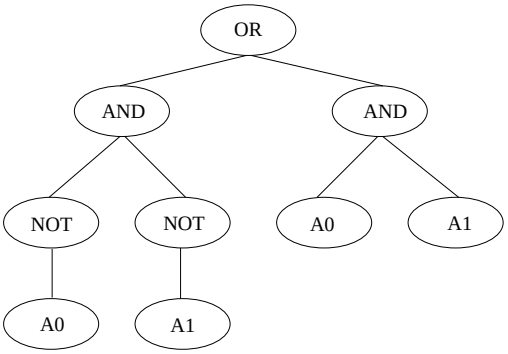

Figure 7. An example of a chromosome used in genetic programming.

The tree-encoding adopted by Koza obviously requires of different alphabets and specialized operators for evolving randomly generated programs until they become 100% valid. Note however, that the basic principles of this technique may be generalized to any other domain and, in fact, genetic programming has been used in a variety of applications [47].

The trees used in genetic programming consist of both functions and terminals. The functions normally adopted are the following [47]:

1. Arithmetic operations (e.g., +, -, ×, ÷)
2. Mathematical functions (e.g., sine, cosine, logarithms, etc.)
3. Boolean Operations (e.g., AND, OR, NOT)
4. Conditionals (IF-THEN-ELSE)
5. Loops (DO-UNTIL)
6. Recursive Functions
7. Any other domain-specific function

Terminals are typically variables or constants, and can be seen as functions that take no arguments. An example of a chromosome that uses the functions F={AND, OR, NOT} and the terminals T={A0, A1} is shown in Figure 7.

Crossover can be applied by numbering the tree nodes corresponding to the two parents chosen (see Figure 8) and (randomly) selecting a point in each of them such that the subtrees below such point are exchanged (see Figure 9, where we assume that the crossover point for the first parent is 2 and for the second is 6). Typically, the sizes of the two parent trees will be different as in the example previously shown. It is also worth noticing that if the crossover point is the root of one of the parent trees, then the whole chromosome will become a subtree of the other parent. This allows the incorporation of subroutines in a program. It is also possible that the roots of both parents are selected as crossover points. Should that be the case, the crossover operator will have no effect and the offspring will be identical to their parents.

Normally, genetic programming implementations impose a limit on the maximum depth that a tree can reach, as to avoid the generation (as a byproduct of crossover and mutation) of trees of very large size that could produce a memory overflow [7].

Mutation in genetic programming takes place through a (random) selection of a certain node tree. The subtree below the chosen node is replaced by another tree which is randomly generated. Figure 10 shows an example of the use of this operator (the mutation point in this example is node 3).

Permutation is an asexual operator used in genetic programming to emulate the effect of the inversion operator adopted with genetic algorithms [34]. This operator reorders the leaves of a subtree placed below a (randomly chosen) node. Its goal is to strengthen the union of allelic combinations with good performance within a chromosome [40].

Figure 11 shows an example of the use of the permutation operator (node 4 was selected in this example). In Figure 11, the symbol '*' indicates multiplication and '%' indicates "protected division", referring to a division operator that keeps our program from generating a system error when the divisor is zero.

In genetic programming is also possible to protect or "encapsulate" a certain subtree which we know to contain a good building block, as to avoid that it is destroyed by the genetic operators. The selected subtree is replaced by a symbolic name that points to the real location of the subtree. Such subtree is separately compiled and linked to the rest of the tree in an analogous way to the external classes of object oriented languages. Figure 12 shows an example of encapsulation in which the right subtree is replaced by the name (**E0**).

Normally, it is also necessary to edit the expressions generated during the evolutionary process as to simplify them. However, the simplication rules to be adopted normally depend on the problem at hand. For example, if the output of our program are Boolean expressions, we could apply rules such as the following:

(AND X X) X
(OR X X) X
(NOT (NOT X)) X

Finally, genetic programming also provides mechanisms to destroy a certain percentage of the population such that we can renovate the chromosomic material after a certain number of generations. This mechanism, called **execution**, is very useful in highly complex domains in which our population may not contain a single feasible individual even after a considerably large number of generations.

3 A More General View of Evolutionary Algorithms

Despite the obvious differences and motivations of each of the aforementioned paradigms, the trend in the last few years has been to decrease the differences among the paradigms and refer (in generic terms) simply to evolutionary algorithms when talking about any of them.

In general, we need the following basic components to implement an EA in order to solve a problem [49]:

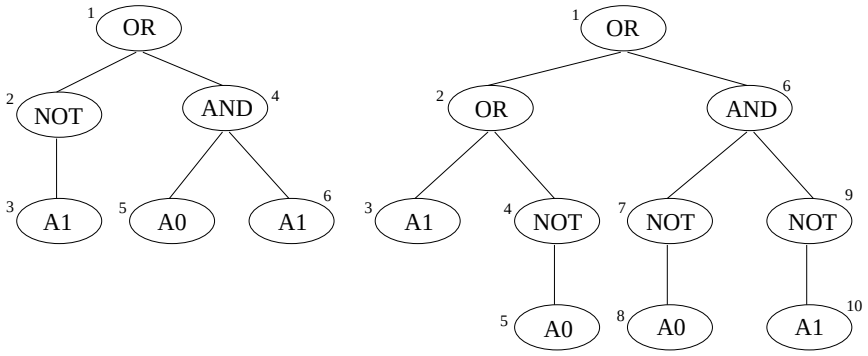

Figure 8. The tree nodes are numbered before applying the crossover operator.

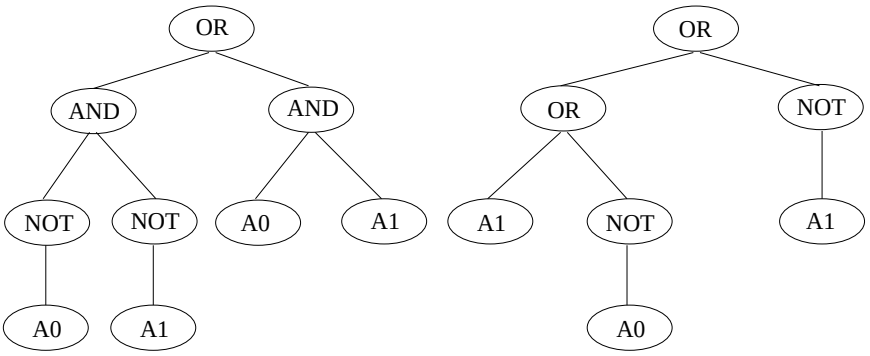

Figure 9. The two offspring generated after applying the crossover operator.

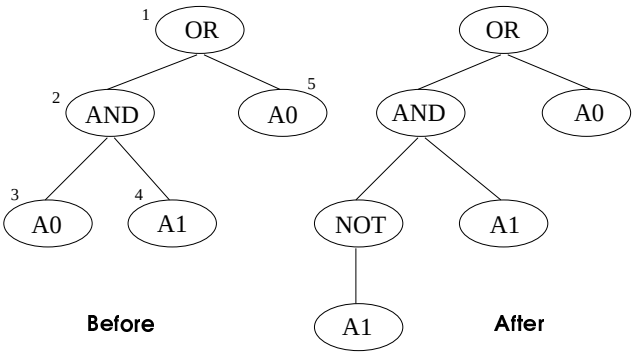

Figure 10. An example of mutation in genetic programming.

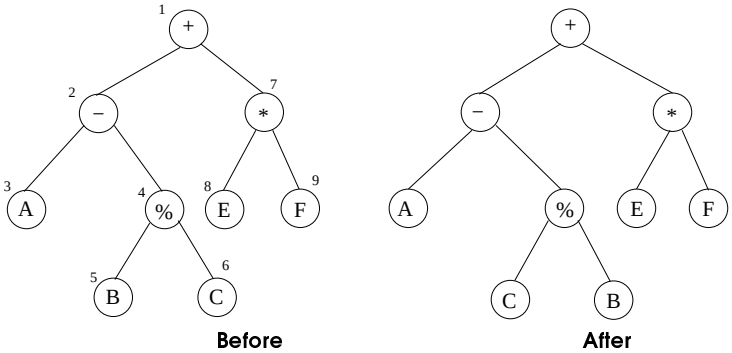

Figure 11. An example of permutation in genetic programming.

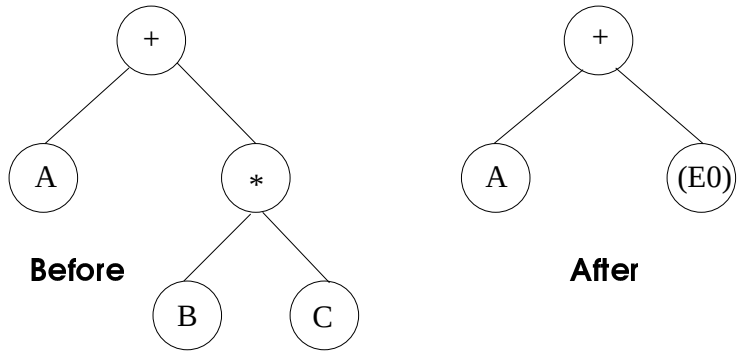

Figure 12. An example of encapsulation in genetic programming.

1. A representation of the potential solutions to the problem. The selection of an appropriate encoding scheme tends to be crucial for the good performance of an EA [60].
2. A way to create an initial population of potential solutions (this is normally done randomly, but deterministic approaches can also be used).
3. An evaluation function that plays the role of the environment, rating solutions in terms of their "fitness". The definition of a good fitness function is also vital for having a good performance.
4. A selection procedure that chooses the parents that will reproduce.
5. Evolutionary operators that alter the composition of children (normally, crossover and mutation).
6. Values for various parameters that the evolutionary algorithm uses (population size, probabilities of applying evolutionary operators, etc.).

EAs differ from traditional search techniques in several ways [11]:

- EAs do not require problem specific knowledge to carry out a search. However, if such knowledge is available, it can be easily incorporated as to make the search more efficient.
- EAs use stochastic instead of deterministic operators and appear to be robust in noisy environments.
- EAs are conceptually simple and easy to implement.
- EAs have a wide applicability.
- EAs are relatively simple to parallelize.
- EAs operate on a population of potential solutions at a time. Thus, they are less susceptible to false attractors (i.e., local optima).

It is worth indicating that the ever-growing popularity of evolutionary algorithms in a variety of application domains tends to be related to their good reputation as "optimizers" (either for single-objective or for multi-objective problems [56, 14]). This is remarkable if we consider that some of them (namely, genetic algorithms) were not originally proposed for that type of application and that their use in optimization tasks has been questioned by some well-established researchers in the evolutionary computation community [43]. Apparently, the reported success of evolutionary algorithms has resulted sufficiently convincing for practitioners and therefore their popularity [4].

4 Some Applications in Biometrics

Since their very inception, several researchers have attempted to use evolutionary algorithms for tasks related to pattern recognition, image processing, classification and machine learning [13, 17, 71, 28, 65, 73, 27]. However, most of this early work was mainly focused on the algorithmic development aspect rather than on an specific application. Thus, for some time, there was relatively little research on the development of evolutionary algorithms for specific biometric applications. This situation has changed in the last few years [63], although the use of other heuristics such as neural networks is still more common than the use of evolutionary algorithms [41].

Next, we will review a few case studies in this area which aim to provide a general picture of the type of research being conducted nowadays.

Fingerprint Identification: Ammar & Tao [1] used genetic algorithms for optimizing the alignment of a pair of fingerprint images. The authors emphasize the importance of the preprocessing stage of the fingerprint identification process. The purpose of this preprocessing stage is to align two regions of interest extracted from each of the two original fingerprints, so that they can be submitted to the decision-

making process. For this application, the authors considered the coordinates of the smaller image to be compared with the larger image (i.e., the fingerprint extracted from a database for possible matching) together with its rotation angle as the decision variables to be encoded (using real numbers) as chromosomes of the genetic algorithm. The correlation coefficient between the smaller image and the larger image was used as the fitness function to be applied to each chromosome. An aspect emphasized in this work is the high computational cost involved in evaluating the fitness function. For a 100×100 image, about 10,000 calculations are required to get the correlation coefficient, which involves a very time-consuming process. In order to overcome this problem, the authors decided to adopt a steady-state genetic algorithm, which tends to replace only a few individuals at each generation (as opposed to traditional genetic algorithms in which the entire population is replaced at each generation). This was more efficient (computationally speaking) for this application and therefore the choice.

For validating their approach, the authors used the NIST Fingerprint Database, which contains 8-bit gray scale images or randomly selected fingerprints. The genetic algorithm was compared with respect to two other approaches: a 2D search algorithm previously proposed by the same authors [2] and a 3D brute-force algorithm. It is worth indicating that these two approaches are basically random search algorithms whose main difference is that the first (2D) ignores the rotation transformation in registration. As a matter of fact, the 3D algorithm tries every possibility and, therefore, it always finds the best correlation.

The results indicated several interesting things. First, the running time of the genetic algorithm did not seem to be sensitive to the size of the search space as opposed to the two other approaches compared. Regarding the quality of the results, the genetic algorithm performed better than the 2D algorithm (the genetic algorithm was, on average, 13% more accurate using the same computational time). Regarding the 3D brute-force approach, the genetic algorithm was able to match its final registration accuracy but using about 1/10th of its computational time.

Face Recognition: Teller and Veloso [68] used genetic programming for face recognition tasks. The authors considered the images to be processed as image signals and thus, they proposed the use of algorithms that analyze signals. For that sake, a system that learns these algorithms through evolution was adopted. The system developed by the authors is called PADO (Parallel Algorithm Discovery and Orchestration) [69]. The learning core of the PADO architecture is genetic programming. The PADO architecture is designed to learn to take signals as input and output correct class labels. Having N classes to choose from, PADO starts by learning N different "systems". Each of these systems is responsible for taking a signal as input and returning a confidence that a certain class C is the correct label. Each system is built out of several programs learned by PADO. It is interesting to note that each of these programs does exactly the same as the system: it takes a signal as input and returns a confidence value regarding the correctness of a given label. PADO performs object recognition by orchestrating the responses of the programs within each system and then does the same with all the systems. Each system is build from the programs that had the best performance when learning to recognize instances of a class (based on their training results). The responses gathered from all the programs are weighted and their weighted average of responses is interpreted as the confidence that the given system has that the signal in question contains an object from a certain class.

PADO evolves programs in a PADO-specific graph structured language [69]. At the beginning of the process, the main population consists of M randomly generated programs (these programs are generated using a grammar for the legal syntax of the language). All programs in this language are constrained by the syntax to return a number that is interpreted as a confidence value between a certain minimum and maximum confidence level. Some "intelligent" crossover and mutation operators are adopted and are co-evolved with the main population [67].

The images used for the validation of the approach were taken from a database from Carnegie-Mellon University. These images consist of head and shoulder shots of 20 different people in an office background. The database has 28 images of each person. So, the authors used a total of 560 images (256×256 8-bit grayscale images were adopted). PADO achieved a 92% recognition rate distinguishing among 20 different face classes.

Handwritten Word Recognition: Morita et al. [53] proposed a methodology for feature selection in unsupervised learning which was applied to handwritten word recognition tasks. The approach is based on a multi-objective evolutionary algorithm (MOEA) called the Nondominated Sorting Genetic Algorithm (NSGA) [64]. Multiobjective optimization problems have 2 or more objective functions (which are normally in conflict with each other). Thus, the notion of optimum is different in this case. The notion of "optimum" that is most commonly adopted is that originally proposed by Francis Ysidro Edgeworth [26] and later generalized by Vilfredo Pareto [57]. MOEAs adopt a different selection scheme from the one used in traditional evolutionary algorithms. In this case, selection schemes that rank the population based on the Pareto optimality of the individuals are commonly adopted [14, 20]. An important property of MOEAs is that they need to be able to maintain several trade-off solutions (called "nondominated") at the same time. Therefore, they normally require special mech-

anisms to maintain diversity and to avoid convergence to a single solution (something that normally occurs with traditional evolutionary algorithms because of stochastic noise).

The goal of the work by Morita et al. [53] was to find a set of nondominated solutions that contained the more discriminant features and the more pertinent number of clusters. The two objectives to be optimized simultaneously were: minimize the number of features and minimize a validity index that measures the quality of clusters. A standard K-Means algorithm was then applied to form the given number of clusters based on the selected features and the number of selected clusters. After performing two experiments on synthetic data sets, the authors adopted a word classifier used in some of their previous work [52]. In their study, a word image was segmented into graphemes, each of which consisted of a correctly segmented, under-segmented, or an over-segmented character. Then, two feature sets were extracted from the sequence of graphemes to feed the classifiers. In order to allow a better assessment of their results, the authors considered only one feature set based on a mixture of segmentation primitives and concavity and contour features. The methodology of the authors consisted of applying the NSGA to obtain a set of nondominated solutions. Then, the incorporation of user's preferences was necessary as to choose only one solution from the set produced. The authors decided to train each nondominated solution obtained. Then they used each classifier in the system and chose the solution that supplied the best word recognition result on the validation set proposed. The results were compared with respect to the use of the traditional methodology previously adopted by the authors for this problem. The authors report that the use of an evolutionary multiobjective optimization approach kept the recognition rates at the same level as the traditional strategy while reducing the time required for training the discrete Hidden Markov Models adopted. This was achieved because the approach reduced the number of features (from 34 to 29) and the number of clusters (from 80 to 36) with respect to the use of a traditional technique.

5 Some Possible Research Directions

Despite the evident popularity of approaches such as neural networks and statistical methods in biometrics, the use of evolutionary algorithms has attracted a growing interest in the last few years. However, there are still many possible research directions that may be worth exploring in this area. Next, we will briefly discuss a few of them:

- **Multiobjective Optimization:** In the previous section, we analyzed an application in which a multiobjective optimization approach was adopted for handwritten character recognition. Due to the multiobjective nature of many problems (i.e., the existence of several conflicting objectives that we aim to optimize at the same time), it is very likely that the use of multiobjective optimization techniques becomes more popular in biometrics. Among the possible techniques to solve multiobjective optimization problems, evolutionary algorithms present several advantages over traditional mathematical programming techniques [50]. For example, evolutionary algorithms have a population-based nature which allows them to generate several elements of the Pareto optimal set in one run. Additionally, evolutionary algorithms do not require an initial search point and are less sensitive to the shape and continuity of the Pareto front [14, 20].

- **Use of Genetic Programming:** As we saw before, the use of tree-encodings in a genetic algorithm (the so-called "genetic programming") is a powerful aid for automated programming. Such type of encoding is also very useful for classification and data mining tasks [33, 70]. Consequently, the use of genetic programming in biometrics, although not widely spread yet, is expected to considerably grow in the next few years. It is worth indicating, however, that the use of a more complex encoding, while allowing tackling more complex problems, also involves a higher computational cost (this applies to multiobjective optimization as well).

- **Use of Alternative Metaheuristics:** In recent years, other biologically-inspired metaheuristics have become increasingly popular in a wide variety of applications [16]. It is expected that several of these approaches are eventually adopted in biometric applications. Representative examples of these new metaheuristics are the following:

  - **Particle Swarm Optimization**: Proposed by Kennedy and Eberhart [44, 46], this metaheuristic simulates the movements of a group (or population) of birds which aim to find food. The approach can be seen as a distributed behavioral algorithm that performs (in its more general version) multidimensional search. In the simulation, the behavior of each individual is affected by either the best local (i.e., within a certain neighborhood) or the best global individual. The approach uses then the concept of population and a measure of performance similar to the fitness value used with evolutionary algorithms. The approach introduces the use of flying potential solutions through hyperspace (used to accelerate convergence) and allows individuals to benefit from their past experiences. This technique has been

successfully used for both continuous nonlinear and discrete binary optimization [25, 45, 46]

– **Artificial Immune Systems**: Computationally speaking, our immune system can be seen as a highly parallel intelligent system that is able to learn and retrieve previous knowledge (i.e., it has "memory") to solve recognition and classification tasks. Due to these interesting features, several researchers have developed computational models of the immune system and have used it for a variety of tasks, including classification and pattern recognition [19, 54, 55].

– **The Ant System**: This is a metaheuristic inspired by colonies of real ants, which deposit a chemical substance on the ground called *pheromone* [22, 15, 24]. This substance influences the behavior of the ants: they tend to take those paths where there is a larger amount of pheromone. Pheromone trails can thus be seen as an indirect communication mechanism among ants. From a computer science perspective, the ant system is a multi-agent system where low level interactions between single agents (i.e., artificial ants) result in a complex behavior of the entire ant colony.

  The ant system was originally proposed for the traveling salesman problem (TSP), and most of the current applications of the algorithm require the problem to be reformulated as one in which the goal is to find the optimal path of a graph. A way to measure the distances between nodes is also required in order to apply the algorithm [23]. Despite this limitation, this approach has been found to be very successful in a variety of combinatorial optimization problems [22, 8].

6 Conclusions

We have provided a general introduction to evolutionary algorithms, discussing their three main paradigms: evolution strategies, evolutionary programming and genetic algorithms. After that, we have analyzed some representative applications of evolutionary algorithms in biometrics, also indicating some potential research paths for future research. As indicated before, it is expected that the use of evolutionary algorithms in biometrics substantially increases within the next few years due to the high potential (and previous evidence of success) of these techniques in a wide variety of applications, including data mining, pattern recognition and classification tasks.

Acknowledgments

The author acknowledges support from CONACyT through project No. 42435-Y.

References

[1] H. T. A. amd Yongyi Tao. Fingerprint registration using genetic algorithms. In *Proceedings of the 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSET'00)*, pages 148–154, Richardson, Texas, March 2000. IEEE Computer Society.

[2] H. Ammar, S. Zeng, and Z. Miao. Parallel processing and fingerprint image comparison. *International Journal on Modelling and Simulation*, 18(2):85–99, 1998.

[3] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

[4] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation.* Institute of Physics Publishing and Oxford University Press, New York, 1997.

[5] J. E. Baker. Adaptive Selection Methods for Genetic Algorithms. In J. J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms*, pages 101–111. Lawrence Erlbaum Associates, Hillsdale, New Jersey, July 1985.

[6] J. E. Baker. Reducing Bias and Inefficiency in the Selection Algorithm. In J. J. Grefenstette, editor, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 14–22. Lawrence Erlbaum Associates, Hillsdale, New Jersey, July 1987.

[7] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Fancone. *Genetic Programming. An Introduction.* Morgan Kaufmann Publishers, San Francisco, California, 1998.

[8] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence. From Natural to Artificial Systems.* Oxford University Press, New York, 1999.

[9] L. B. Booker. *Intelligent Behavior as an Adaptation to the Task Environment.* PhD thesis, Logic of Computers Group, University of Michigan, Ann Arbor, Michigan, 1982.

[10] A. Brindle. *Genetic Algorithms for Function Optimization*. PhD thesis, Department of Computer Science, University of Alberta, Edmonton, Alberta, 1981.

[11] B. P. Buckles and F. E. Petry, editors. *Genetic Algorithms*. Technology Series. IEEE Computer Society Press, 1992.

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

[13] D. Cavicchio. *Adaptive Search Using Simulated Evolution*. PhD thesis, University of Michigan, Ann Arbor, Michigan, 1970.

[14] C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, May 2002. ISBN 0-3064-6762-3.

[15] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed Optimization by Ant Colonies. In F. J. Varela and P. Bourgine, editors, *Proceedings of the First European Conference on Artificial Life*, pages 134–142. MIT Press, Cambridge, MA, 1992.

[16] D. Corne, M. Dorigo, and F. Glover, editors. *New Ideas in Optimization*. McGraw-Hill, London, 1999.

[17] F. N. Cornett. An Application of Evolutionary Programming to Pattern Recognition. Master's thesis, New Mexico State University, Las Cruces, New Mexico, 1972.

[18] C. R. Darwin. *The Variation of Animals and Plants under Domestication*. Murray, London, second edition, 1882.

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

[20] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, Chichester, UK, 2001. ISBN 0-471-87339-X.

[21] K. Deb and D. E. Goldberg. An Investigation of Niche and Species Formation in Genetic Function Optimization. In J. D. 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.

[22] M. Dorigo and G. D. Caro. The Ant Colony Optimization Meta-Heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32, London, 1999. McGraw-Hill.

[23] M. Dorigo, V. Maniezzo, and A. Colorni. Positive Feedback as a Search Strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991.

[24] M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 26(1):29–41, 1996.

[25] R. Eberhart and Y. Shi. Comparison between Genetic Algorithms and Particle Swarm Optimization. In V. W. Porto, N. Saravanan, D. Waagen, and A. Eibe, editors, *Proceedings of the Seventh Annual Conference on Evolutionary Programming*, pages 611–619. Springer-Verlag, March 1998.

[26] F. Y. Edgeworth. *Mathematical Physics*. P. Keagan, London, England, 1881.

[27] A. C. Englander. Machine Learning of Visual Recognition Using Genetic Algorithms. In J. J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, pages 197–201, Hillsdale, New Jersey, July 1985. Lawrence Erlbraum Associates, Publishers.

[28] J. Fitzpatrick, J. Grefenstette, and D. Van Gutch. Image registration by genetic search. In *Proceedings of the IEEE Southeast Conference*, pages 460–464, 1984.

[29] D. B. Fogel. *Evolutionary Computation. Toward a New Philosophy of Machine Intelligence*. The Institute of Electrical and Electronic Engineers, New York, 1995.

[30] D. B. Fogel, editor. *Evolutionary Computation. The Fossil Record. Selected Readings on the History of Evolutionary Algorithms*. The Institute of Electrical and Electronic Engineers, New York, 1998.

[31] L. J. Fogel. *Artificial Intelligence through Simulated Evolution*. John Wiley, New York, 1966.

[32] L. J. Fogel. *Artificial Intelligence through Simulated Evolution. Forty Years of Evolutionary Programming*. John Wiley & Sons, Inc., New York, 1999.

[33] A. A. Freitas. A Genetic Programming Framework for Two Data Mining Tasks: Classification and Generalized Rule Induction. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, editors, *Proceedings of the Second Annual Conference on Genetic Programming*, pages 96–101, San Francisco, California, July 1997. Morgan Kaufmann Publishers.

[34] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Co., Reading, Massachusetts, 1989.

[35] D. E. Goldberg and K. Deb. A comparison of selection schemes used in genetic algorithms. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, San Mateo, California, 1991.

[36] J. J. Grefenstette and J. E. Baker. How Genetic Algorithms work: A critical look at implicit parallelism. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 20–27, San Mateo, California, June 1989. Morgan Kaufmann Publishers.

[37] J. Heitkoetter and D. Beasley. The hitch-hiker's guide to evolutionary computation (faq in comp.ai.genetic). USENET, sep 1995. (Version 3.3).

[38] J. H. Holland. Concerning efficient adaptive systems. In M. C. Yovits, G. T. Jacobi, and G. D. Goldstein, editors, *Self-Organizing Systems—1962*, pages 215–230. Spartan Books, Washington, D.C., 1962.

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

[40] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, 1975.

[41] K. Huang and H. Yan. Off-Line Signature Verification by a Neural Network Classifier. In *Proceedings of the 17th Australian Conference on Neural Networks*, pages 190–194, Canberra, Australia, 1996. Australian National University.

[42] A. K. D. Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

[43] K. A. D. Jong. Genetic Algorithms are NOT Function Optimizers. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 5–17. Morgan Kaufmann Publishers, San Mateo, California, 1993.

[44] J. Kennedy and R. C. Eberhart. Particle Swarm Optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, pages 1942–1948, Piscataway, New Jersey, 1995. IEEE Service Center.

[45] J. Kennedy and R. C. Eberhart. A Discrete Binary Version of the Particle Swarm Algorithm. In *Proceedings of the 1997 IEEE Conference on Systems, Man, and Cybernetics*, pages 4104–4109, Piscataway, New Jersey, 1997. IEEE Service Center.

[46] J. Kennedy and R. C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco, California, 2001.

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

[48] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, second edition, 1992.

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

[50] K. M. Miettinen. *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, Boston, Massachusetts, 1998.

[51] M. Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, Cambridge, Massachusetts, 1996.

[52] M. Morita, R. Sabourin, F. Bortolozzi, and C. Suen. Segmentation and recognition of handwritten dates. In *Proceedings of the Eighth International Workshop on Frontiers of Handwriting Recognition (IWFHR'02)*, pages 105–110, Ontario, Canada, August 2002. IEEE Computer Society.

[53] M. Morita, R. Sabourin, F. Bortolozzi, and C. Suen. Unsupervised feature selection using multi-objective genetic algorithm for handwritten word recognition. In *Proceedings of the 7th International Conference on Document Analysis and Recognition (ICDAR'2003)*, pages 666–670, Edinburgh, Scotland, August 2003.

[54] L. Nunes de Castro and J. Timmis. *Artificial Immnue System: A New Computational Intelligence Approach*. Springer Verlang, Great Britain, September 2002. ISBN 1-8523-594-7.

[55] L. Nunes de Castro and F. J. Von Zuben. Learning and Optimization Using the Clonal Selection Principle. *IEEE Transactions on Evolutionary Computation*, 6(3):239–251, 2002.

[56] A. Osyczka. *Evolutionary Algorithms for Single and Multicriteria Design Optimization*. Physica Verlag, Germany, 2002. ISBN 3-7908-1418-0.

[57] V. Pareto. *Cours D'Economie Politique*, volume I and II. F. Rouge, Lausanne, 1896.

[58] S. S. Rao. *Engineering Optimization. Theory and Practice*. John Wiley & Sons, Inc., third edition, 1996.

[59] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann–Holzboog, Stuttgart, Germany, 1973.

[60] F. Rothlauf. *Representations for Genetic and Evolutionary Algorithms*. Physica-Verlag, New York, 2002.

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

[62] H.-P. Schwefel. *Numerical Optimization of Computer Models*. Wiley, Chichester, UK, 1981.

[63] J. Soldek, V. Shmerko, P. Phillips, G. Kukharev, W. Rogers, and S. Yanushkevich. Image Analysis and Pattern Recognition in Biometric Technologies. In *Proceedings of the International Conference on the Biometrics: Fraud Prevention, Enhanced Service*, pages 270–286, Las Vegas, Nevada, 1997.

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

[65] I. Stadnyk. Schema Recombination in a Pattern Recognition Problem. In J. J. Grefenstette, editor, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 27–35, Hillsdale, New Jersey, July 1987. Lawrence Erlbraum Associates, Publishers.

[66] G. Syswerda. Uniform Crossover in Genetic Algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9, San Mateo, California, 1989. Morgan Kaufmann Publishers.

[67] A. Teller. Evolving Programmers: The Co-evolution of Intelligent Recombination Operators. In P. J. Angeline and K. E. K. Jr., editors, *Advances in Genetic Programming. Volume 2*, pages 45–68. The MIT Press, Cambridge, Massachusetts, 1996.

[68] A. Teller and M. Veloso. Algorithm Evolution for Face Recognition: What Makes a Picture Difficult. In *International Conference on Evolutionary Computation (ICEC'95)*, pages 608–613, Perth, Australia, 1995. IEEE.

[69] A. Teller and M. Veloso. PADO: A New Learning Architecture for Object Recognition. In K. Ikeuchi and M. Veloso, editors, *Symbolic Visual Learning*, pages 81–116. Oxford University Press, 1996.

[70] A. Teredasai, J. Park, and V. Govindaraju. Active Handwritten Character Recognition Using Genetic Programming. In J. Miller, M. Tomassini, P. L. Lanzi, C. Ryan, A. G. Tettamanzi, and W. B. Langdon, editors, *Genetic Programming. 4th European Conference, EuroGP 2001*, pages 371–379, Lake Como, Italy, April 2001. Springer.

[71] R. Trellue. The Recognition of Handprinted Characters Through Evolutionary Programming. Master's thesis, New Mexico State University, Las Cruces, New Mexico, 1973.

[72] D. Whitley. The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121. Morgan Kaufmann Publishers, San Mateo, California, July 1989.

[73] S. W. Wilson. Adaptive "Cortical" Pattern Recognition. In J. J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, pages 188–196, Hillsdale, New Jersey, July 1985. Lawrence Erlbraum Associates, Publishers.

[74] D. D. Zhang. *Automated Biometrics: Technologies and Systems*. Kluwer Academic Publishers, May 2000.

Carlos A. Coello Coello received the B.Sc. degree in civil engineering from the Universidad Autónoma de Chiapas, México, and the M.Sc. and the PhD degrees in computer science from Tulane University, USA, in 1991, 1993, and 1996, respectively.

He is currently an associate professor (CINVESTAV-3B Researcher) at the electrical engineering department of CINVESTAV-IPN, in Mexico City, México. Dr. Coello has authored and co-authored over 100 technical papers and several book chapters. He has also co-authored the book *Evolutionary Algorithms for Solving Multi-Objective Problems* (Kluwer Academic Publishers, 2002). Additionally, Dr. Coello has served in the program committees of over 30 international conferences and has been technical reviewer for over 30 international journals including the *IEEE Transactions on Evolutionary Computation* in which he also serves as Associate Editor. He also chairs the *Task Force on*

Multi-Objective Evolutionary Algorithms of the IEEE Neural Networks Society. He is a member of the IEEE, the ACM and the Mexican Academy of Sciences.

His major research interests are: evolutionary multi-objective optimization, constraint-handling techniques for evolutionary algorithms, and evolvable hardware.