An Introduction to Evolutionary Algorithms and Their Applications

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 most representative applications. 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 real-world problems are analyzed. In the final part of the paper, some of the current research directions in this area 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 [1]. 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 [2–4].

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 [5]. Biometric applications involve several complex problems. For example, many current biometric applications are closely related to pattern recognition and image analysis [6]. 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 [4, 2, 7].

The remainder of this paper is organized as follows. Section 2 provides some basic concepts related to evolutionary algorithms. Section 3 discusses a few representative case studies of applications of evolutionary algorithms in biometrics. After that, we provide some possible future research directions in Section 4 and our conclusions in Section 5.

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* [8]. 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) [9], 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 [3], evolutionary programming [10], and genetic algorithms [11]. Additionally, some authors consider genetic programming [12] 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 [13]. 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 [14]. 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 [15] 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 [3]). $\sigma(t)$ 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 [3]:

- 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 [16, 10].

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 [2]).

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

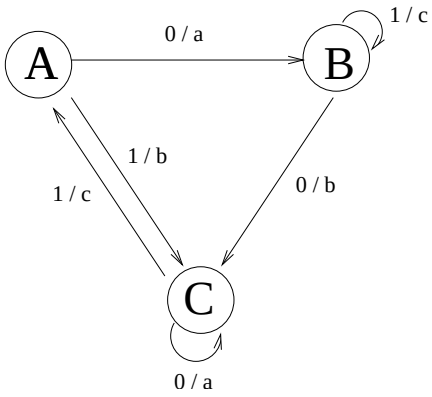

Fig. 1. Finite automaton with 3 states. Symbols to the left of "/" are input symbols. Symbols to the right of "/" are output symbols. The initial state is A.

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 [4]:

- 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 [17, 18]. 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).

The basic operation of a Genetic Algorithm is illustrated in the following segment of pseudo-code [19]:

```
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.

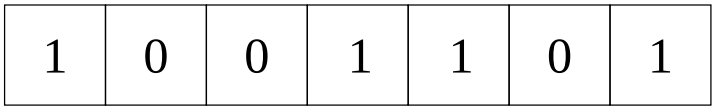

Fig. 2. Example of the binary encoding traditionally adopted with the genetic algorithm.

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 [2]. Holland [11] 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 [20, 21].

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 [22]:

– *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 [23], *stochastic remainder* selection [24, 25], and *stochastic universal* selection [26, 27].

- *Ranking Selection:* In this scheme, proposed by Baker [28] 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 [29], 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 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 [30].

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 [31] 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 [31], in general, one-point crossover seems to be a bad choice, but there is no clear winner between two-point and uniform crossover [32].

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

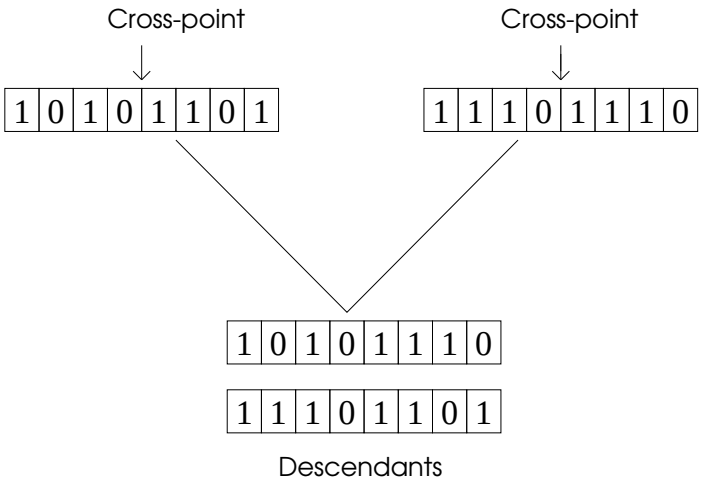

Fig. 3. Use of a single-point crossover between two chromosomes. Notice that each pair of chromosomes produces two descendants for the next generation. The cross-point may be located at the string boundaries, in which case the crossover has no effect and the parents remain intact for the next generation.

theoretical perspective, it assures that—given any population—the entire search space is connected [19].

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 [2]:

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

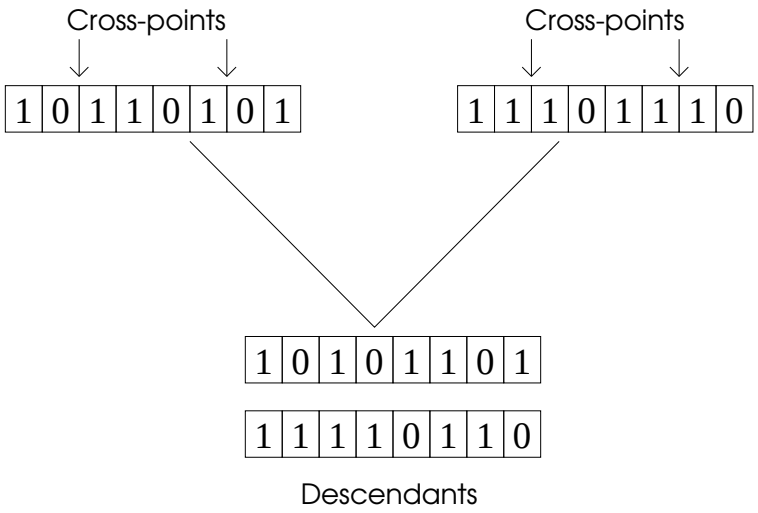

Fig. 4. Use of a two-point crossover between two chromosomes. In this case the genes at the extremes are kept, and those in the middle part are exchanged. If one of the two cross-points happens to be at the string boundaries, a single-point crossover will be performed, and if both are at the string boundaries,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

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 [4]. 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 [12] 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.

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 [12].

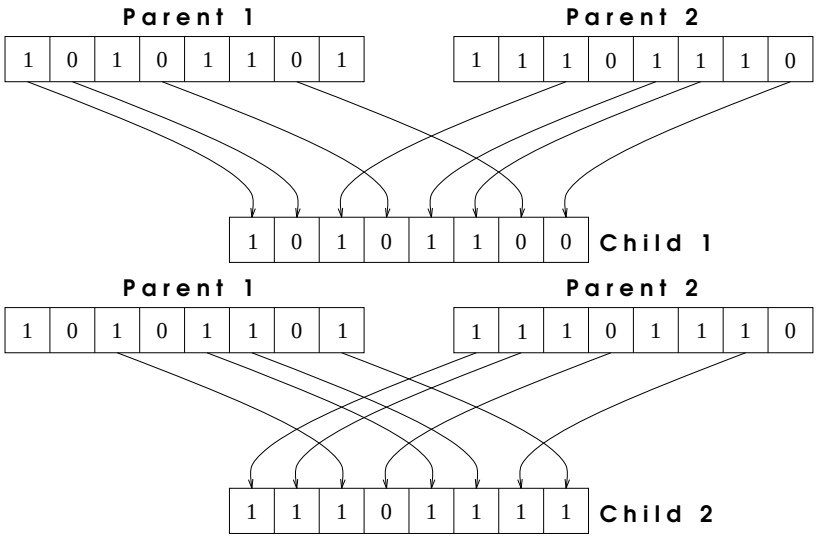

Fig. 5. Use of 0.5-uniform crossover (using 50% probability) between two chromosomes. Notice how half of the genes of each parent goes to each of the two children. First, the bits to be copied from each parent are selected randomly using the probability desired, and after the first child is generated,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

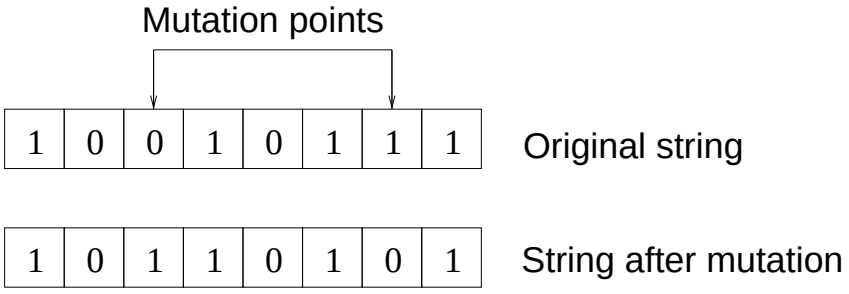

Fig. 6. An Example of mutation using binary representation.

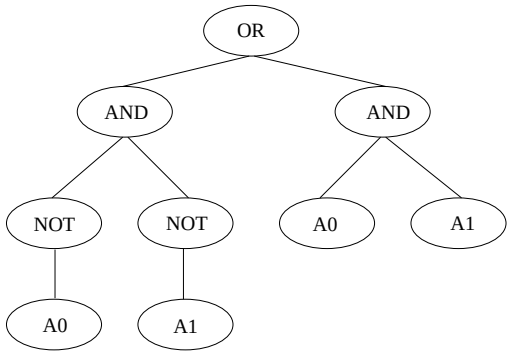

Fig. 7. An example of a chromosome used in genetic programming.

The trees used in genetic programming consist of both functions and terminals. The functions normally adopted are the following [12]:

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

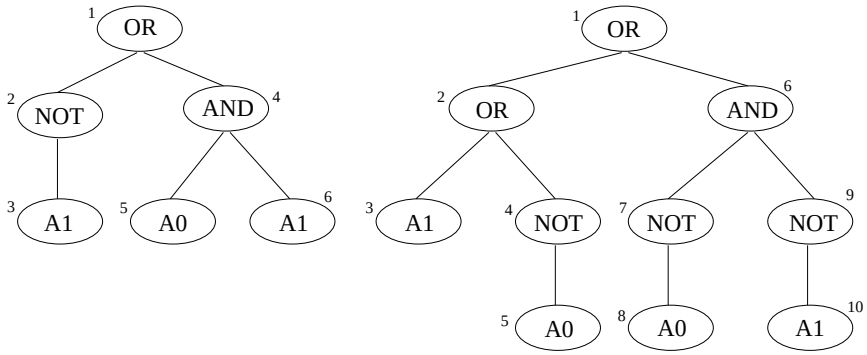

Fig. 8. The tree nodes are numbered before applying the crossover operator.

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

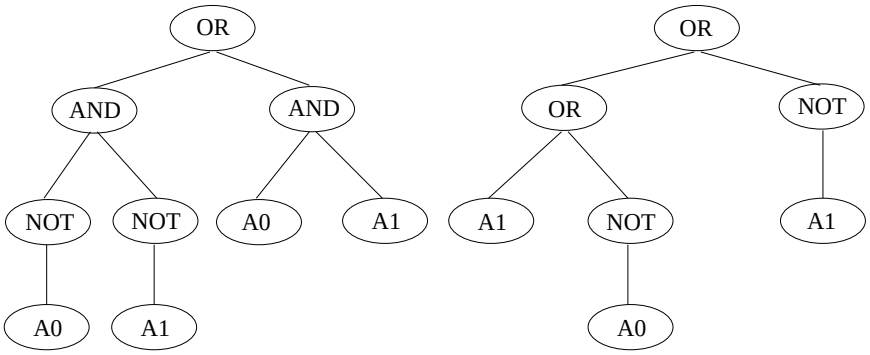

Fig. 9. The two offspring generated after applying the crossover operator.

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 [33].

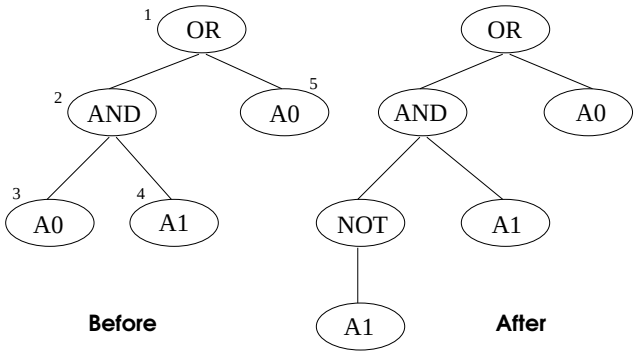

Fig. 10. An example of mutation in genetic programming.

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).

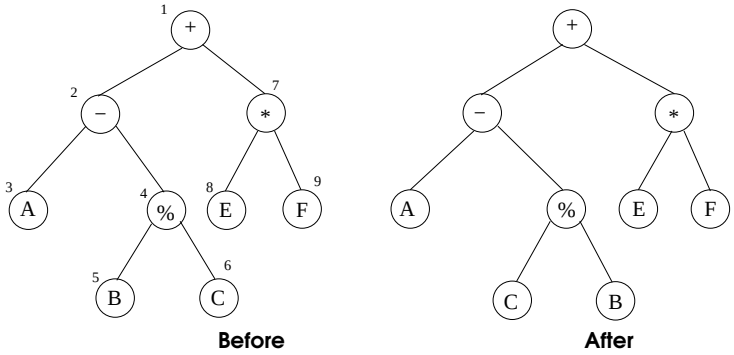

Fig. 11. An example of permutation in genetic programming.

Permutation is an asexual operator used in genetic programming to emulate the effect of the inversion operator adopted with genetic algorithms [2]. 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 [11].

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.

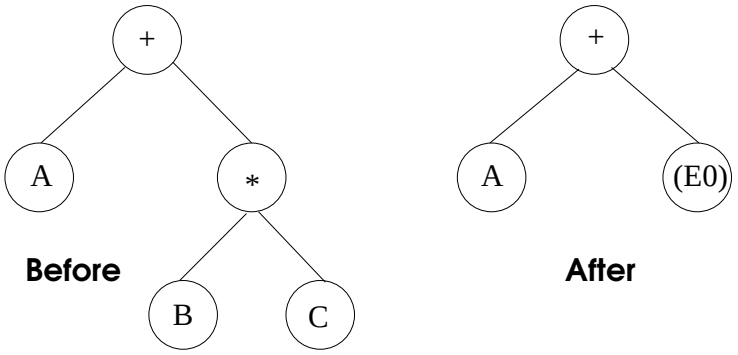

Fig. 12. An example of encapsulation in genetic programming.

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.

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.

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 [34, 35]). 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 [36]. Apparently, the reported success of evolutionary algorithms has resulted sufficiently convincing for practitioners and therefore their popularity [7].

3 A Few Real-World Applications

Next, we will briefly review a few interesting real-world applications of evolutionary algorithms that have been reported in the specialized literature. Note that our coverage of such applications will be shallow, but the interested reader is referred to further references in case of having interest in any of them.

3.1 Improved Satellite Constellation Design and Optimization

Improving satellite constellation design is of great interest to any users of satellite communications (e.g. cell phones, television), location (e.g. global positioning system) and/or observation (e.g. weather). Many of todays satellite constellation designs rely on the "Walker Constellations," a series of designs developed in 1970, which have rarely been improved upon. These constellations make use of symmetric constellations with circular orbits. Dr. William Crossley (from Purdue University) has used genetic algorithms to search the constellation design space, producing constellation designs not previously envisioned but with performance equal to or greater than comparable Walker or

"streets of coverage" constellations. The designs produced by the genetic algorithm took some seasoned satellite engineers by surprise. Dr. Crossley is now experimenting with multiobjective genetic algorithms and has been able to generate constellation designs that outperformed constellations that had been under development for several months. The genetic algorithm used in this case, required just a few days to produce such designs. For more details, see [37].

3.2 Telecommunication Network Design

Prof. Benjam´ın Bar´an and his research group in Paraguay used parallel multi-objective evolutionary algorithms to design an optimal telecommunication network in which the objectives were to minimize cost and maximize performance. The network was modelled using graphs and encoded adopting integers. Reliability was estimated using Monte Carlo simulations, and the cost of each configuration produced was computed by adding up the costs of every link added to the topology. The authors also considered a minimum acceptable reliability. In their study, the authors used a well-established problem in the area to validate their approach. The results produced by the evolutionary algorithms adopted outperformed the best solution previously reported for the problem used as a reference. For more information, see [38].

3.3 Design of Mobile Telecommunication Networks

Meunier and his colleagues used a multi-objective evolutionary algorithm to design a mobile telecommunication network that consists of positioning base stations in potential sites such that 3 objectives are fulfilled: minimize the number of sites used, maximize the amount of traffic held by the network and minimize the interferences. The authors also considered two constraints: (1) all the service test points must me covered with a minimum radio field value that must be greater than the receiver sensitivity threshold of the mobile and (2) the cellular network must be able to ensure the communication continuity from the starting cell to the target cell, when a mobile is moving toward a new cell. This is a combinatorial optimization problem with a high computational cost associated with the evaluation of each potential solution. The authors used parallelism and tested their genetic algorithm on a large and realistic highway area generated by the France Telecom's research laboratory (CNET). Results indicated that the use of evolutionary algorithms is a very promising alternative in this type of problem. For further details, see [39].

4 Some Current Research Trends

In recent years, other biologically-inspired metaheuristics have become increasingly popular in a wide variety of applications [40]. 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 [41, 42], 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 [43, 44, 42]
– **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 [45–47].
– **The Ant System**: This is a metaheuristic inspired by colonies of real ants, which deposit a chemical substance on the ground called *pheromone* [48–50]. 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 [51]. Despite this limitation, this approach has been found to be very successful in a variety of combinatorial optimization problems [48, 52].
– **Cultural Algorithms**: Cultural algorithms were developed by Robert G. Reynolds as a complement to the metaphor used by evolutionary algorithms, which had focused mainly on genetic and natural selection concepts [53]. Cultural algorithms are based on some theories originated in sociology and archaeology which try to model cultural evolution. Such theories indicate that cultural evolution can be seen as an inheritance process operating at two levels: (1) a micro-evolutionary level, which consists of the genetic material that an offspring inherits from its parents, and (2) a macro-evolutionary level, which consists of the knowledge acquired by individuals through generations. This knowledge, once encoded and stored, is used to guide the behavior of the individuals that belong to a certain population [54, 55]. Culture can be seen as a set of ideological phenomena shared by a population. Through these phenomena, an individual can interpret its experiences and decide its behavior. In these models, we can clearly appreciate the part of the system that is shared by the population: the knowledge, acquired by members of a society, but encoded in such a way that such knowledge can be accessed by every other member

of the society. And then there is an individual part, which consists of the interpretation of such knowledge encoded in the form of symbols. This interpretation will produce new behaviors as a consequence of the assimilation of the corresponding knowledge acquired combined with the experiences lived by the individual itself. Reynolds attempts to capture this double inheritance phenomenon through his proposal of cultural algorithms [53]. The main goal of such algorithms is to increase the learning or convergence rates of an evolutionary algorithm such that the system can respond better to a wide variety of problems [56]. Cultural algorithms operate in two spaces. First, we have the population space, which consists of (as in all evolutionary algorithms) a set of individuals. Each individual has a set of independent features that are used to determine its fitness. Through time, such individuals can be replaced by some of their descendants, which are obtained from a set of operators applied to the population. The second space is the belief space, which is where we store the knowledge acquired by individuals through generations. The information contained in this space must be accessible to each individual, so that they can use it to modify their behavior. In order to join the two spaces, it is necessary to provide a communication link, which dictates the rules regarding the type of information that must be exchanged between the two spaces.

5 Conclusions

Evolutionary algorithms are a viable alternative to solve complex, real-world problems for which no other approach provides acceptable results in a reasonably short time. Due to their heuristic nature, evolutionary algorithms cannot guarantee that the solution that they find is the global optimum. However, there is overwhelming evidence regarding the effectiveness of evolutionary algorithms to solve real-world problems when compared with other (either deterministic or heuristic) approaches.

Acknowledgments

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

References

1. Cannon, W.D.: The Wisdom of the Body. Norton and Company, New York (1932)
2. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Publishing Co., Reading, Massachusetts (1989)
3. Schwefel, H.P.: Numerical Optimization of Computer Models. Wiley, Chichester, UK (1981)
4. Fogel, D.B.: Evolutionary Computation. Toward a New Philosophy of Machine Intelligence. The Institute of Electrical and Electronic Engineers, New York (1995)
5. Zhang, D.D.: Automated Biometrics: Technologies and Systems. Kluwer Academic Publishers (2000)
6. Soldek, J., Shmerko, V., Phillips, P., Kukharev, G., Rogers, W., Yanushkevich, S.: Image Analysis and Pattern Recognition in Biometric Technologies. In: Proceedings of the International Conference on the Biometrics: Fraud Prevention, Enhanced Service, Las Vegas, Nevada (1997) 270–286

7. Bäck, T., Fogel, D.B., Michalewicz, Z., eds.: Handbook of Evolutionary Computation. Institute of Physics Publishing and Oxford University Press, New York (1997)
8. Darwin, C.R.: The Variation of Animals and Plants under Domestication. Second edn. Murray, London (1882)
9. Bäck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, New York (1996)
10. Fogel, L.J.: Artificial Intelligence through Simulated Evolution. Forty Years of Evolutionary Programming. John Wiley & Sons, Inc., New York (1999)
11. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, Michigan (1975)
12. Koza, J.R.: Genetic Programming. On the Programming of Computers by Means of Natural Selection. The MIT Press, Cambridge, Massachusetts (1992)
13. Rao, S.S.: Engineering Optimization. Theory and Practice. Third edn. John Wiley & Sons, Inc. (1996)
14. Fogel, D.B., ed.: Evolutionary Computation. The Fossil Record. Selected Readings on the History of Evolutionary Algorithms. The Institute of Electrical and Electronic Engineers, New York (1998)
15. Rechenberg, I.: Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann–Holzboog, Stuttgart, Germany (1973)
16. Fogel, L.J.: Artificial Intelligence through Simulated Evolution. John Wiley, New York (1966)
17. Holland, J.H.: Concerning efficient adaptive systems. In Yovits, M.C., Jacobi, G.T., Goldstein, G.D., eds.: Self-Organizing Systems—1962. Spartan Books, Washington, D.C. (1962) 215–230
18. Holland, J.H.: Outline for a logical theory of adaptive systems. Journal of the Association for Computing Machinery **9** (1962) 297–314
19. Buckles, B.P., Petry, F.E., eds.: Genetic Algorithms. Technology Series. IEEE Computer Society Press (1992)
20. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Third edn. Springer-Verlag, New York (1996)
21. Rothlauf, F.: Representations for Genetic and Evolutionary Algorithms. Physica-Verlag, New York (2002)
22. Goldberg, D.E., Deb, K.: A comparison of selection schemes used in genetic algorithms. In Rawlins, G.J.E., ed.: Foundations of Genetic Algorithms. Morgan Kaufmann, San Mateo, California (1991) 69–93
23. Jong, A.K.D.: An Analysis of the Behavior of a Class of Genetic Adaptive Systems. PhD thesis, University of Michigan (1975)
24. Booker, L.B.: Intelligent Behavior as an Adaptation to the Task Environment. PhD thesis, Logic of Computers Group, University of Michigan, Ann Arbor, Michigan (1982)
25. Brindle, A.: Genetic Algorithms for Function Optimization. PhD thesis, Department of Computer Science, University of Alberta, Edmonton, Alberta (1981)
26. Baker, J.E.: Reducing Bias and Inefficiency in the Selection Algorithm. In Grefenstette, J.J., ed.: Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms. Lawrence Erlbaum Associates, Hillsdale, New Jersey (1987) 14–22
27. Grefenstette, J.J., Baker, J.E.: How Genetic Algorithms work: A critical look at implicit parallelism. In Schaffer, J.D., ed.: Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, California, Morgan Kaufmann Publishers (1989) 20–27
28. Baker, J.E.: Adaptive Selection Methods for Genetic Algorithms. In Grefenstette, J.J., ed.: Proceedings of the First International Conference on Genetic Algorithms. Lawrence Erlbaum Associates, Hillsdale, New Jersey (1985) 101–111

29. Whitley, D.: The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. In Schaffer, J.D., ed.: Proceedings of the Third International Conference on Genetic Algorithms. Morgan Kaufmann Publishers, San Mateo, California (1989) 116–121
30. Mitchell, M.: An Introduction to Genetic Algorithms. The MIT Press, Cambridge, Massachusetts (1996)
31. Syswerda, G.: Uniform Crossover in Genetic Algorithms. In Schaffer, J.D., ed.: Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, California, Morgan Kaufmann Publishers (1989) 2–9
32. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Second edn. Springer-Verlag (1992)
33. Banzhaf, W., Nordin, P., Keller, R.E., Fancone, F.D.: Genetic Programming. An Introduction. Morgan Kaufmann Publishers, San Francisco, California (1998)
34. Osyczka, A.: Evolutionary Algorithms for Single and Multicriteria Design Optimization. Physica Verlag, Germany (2002) ISBN 3-7908-1418-0.
35. Coello Coello, C.A., Van Veldhuizen, D.A., Lamont, G.B.: Evolutionary Algorithms for Solving Multi-Objective Problems. Kluwer Academic Publishers, New York (2002) ISBN 0-3064-6762-3.
36. Jong, K.A.D.: Genetic Algorithms are NOT Function Optimizers. In Whitley, L.D., ed.: Foundations of Genetic Algorithms 2. Morgan Kaufmann Publishers, San Mateo, California (1993) 5–17
37. Ely, T., Crossley, W., Williams, E.: Satellite Constellation Design for Zonal Coverage Using Genetic Algorithms. Journal of the Astronautical Sciences **47** (1999) 207–228
38. Duarte Flores, S., Barán Cegla, B., Benítez Cáceres, D.: Telecommunication network design with parallel multi-objective evolutionary algorithms. In: Applications, Technologies, Architectures,and Protocols for Computer Communication. Proceedings of the 2003 IFIP/ACM Latin America conference on Towards a Latin American agenda for network research, La Paz, Bolivia, ACM Press (2003) 1–11
39. Meunier, H., Talbi, E.G., Reininger, P.: A Multiobjective Genetic Algorithm for Radio Network Optimization. In: 2000 Congress on Evolutionary Computation. Volume 1., Piscataway, New Jersey, IEEE Service Center (2000) 317–324
40. Corne, D., Dorigo, M., Glover, F., eds.: New Ideas in Optimization. McGraw-Hill, London (1999)
41. Kennedy, J., Eberhart, R.C.: Particle Swarm Optimization. In: Proceedings of the 1995 IEEE International Conference on Neural Networks, Piscataway, New Jersey, IEEE Service Center (1995) 1942–1948
42. Kennedy, J., Eberhart, R.C.: Swarm Intelligence. Morgan Kaufmann Publishers, San Francisco, California (2001)
43. Eberhart, R., Shi, Y.: Comparison between Genetic Algorithms and Particle Swarm Optimization. In Porto, V.W., Saravanan, N., Waagen, D., Eibe, A., eds.: Proceedings of the Seventh Annual Conference on Evolutionary Programming, Springer-Verlag (1998) 611–619
44. Kennedy, J., Eberhart, R.C.: A Discrete Binary Version of the Particle Swarm Algorithm. In: Proceedings of the 1997 IEEE Conference on Systems, Man, and Cybernetics, Piscataway, New Jersey, IEEE Service Center (1997) 4104–4109
45. Dasgupta, D., ed.: Artificial Immune Systems and Their Applications. Springer-Verlag, Berlin (1999)
46. Nunes de Castro, L., Timmis, J.: Artificial Immnue System: A New Computational Intelligence Approach. Springer Verlang, Great Britain (2002) ISBN 1-8523-594-7.
47. Nunes de Castro, L., Von Zuben, F.J.: Learning and Optimization Using the Clonal Selection Principle. IEEE Transactions on Evolutionary Computation **6** (2002) 239–251

48. Dorigo, M., Caro, G.D.: The Ant Colony Optimization Meta-Heuristic. In Corne, D., Dorigo, M., Glover, F., eds.: New Ideas in Optimization, London, McGraw-Hill (1999) 11–32
49. Colorni, A., Dorigo, M., Maniezzo, V.: Distributed Optimization by Ant Colonies. In Varela, F.J., Bourgine, P., eds.: Proceedings of the First European Conference on Artificial Life, MIT Press, Cambridge, MA (1992) 134–142
50. Dorigo, M., Maniezzo, V., Colorni, A.: The Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics – Part B **26** (1996) 29–41
51. Dorigo, M., Maniezzo, V., Colorni, A.: Positive Feedback as a Search Strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy (1991)
52. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence. From Natural to Artificial Systems. Oxford University Press, New York (1999)
53. Reynolds, R.G.: An Introduction to Cultural Algorithms. In Sebald, A.V., , Fogel, L.J., eds.: Proceedings of the Third Annual Conference on Evolutionary Programming. World Scientific, River Edge, New Jersey (1994) 131–139
54. Renfrew, A.C.: Dynamic Modeling in Archaeology: What, When, and Where? In van der Leeuw, S.E., ed.: Dynamical Modeling and the Study of Change in Archaelogy. Edinburgh University Press, Edinburgh, Scotland (1994)
55. Durham, W.H.: Co-evolution: Genes, Culture, and Human Diversity. Stanford University Press, Stanford, California (1994)
56. Franklin, B., Bergerman, M.: Cultural algorithms: Concepts and experiments. In: Proceedings of the 2000 Congress on Evolutionary Computation, Piscataway, New Jersey, IEEE Service Center (2000) 1245–1251