A Review of Applications of Evolutionary Algorithms in Pattern Recognition

Luis Gerardo de la Fraga and Carlos A. Coello Coello*

CINVESTAV-IPN (Evolutionary Computation Group)
Departamento de Computación
Av. IPN No. 2508, Col. San Pedro Zacatenco
Mexico, D.F. 07360, MEXICO
{fraga|ccoello}@cs.cinvestav.mx

Abstract This chapter presents a review of some of the most representative work regarding techniques and applications of evolutionary algorithms in pattern recognition. Evolutionary algorithms are a set of metaheuristics inspired on Darwin's "survival of the fittest" principle which are stochastic in nature. Evolutionary algorithms present several advantages over traditional search and classification techniques, since they require less domain-specific information, are easy to use and operate on a set of solutions (the so-called population). Such advantages have made them very popular within pattern recognition (as well as in other domains) as will be seen in the review of applications presented in this chapter.

Key words: evolutionary algorithms, pattern recognition, genetic algorithms, differential evolution, genetic programming

1 Introduction

We will start by providing a very general description of a pattern recognition process [1, 2] in order to make this chapter self-contained.

The pattern recognition process consists of the three stages shown in Fig. 1: (1) segmentation, (2) feature selection and (3) classification. In Fig. 1, the input is a set of pixels (i.e., an image), but other types of input data are also possible (e.g., three dimensional scanned points of a human face).

Segmentation is the partition of an input image, or data, into its constituent parts or objects. It is known that, in general, automatic segmentation is a very difficult task. The output of the segmentation stage is usually another image with raw pixel data, constituting either the boundary of a region or all the points in the region itself.

* The second author is also with the UMI LAFMIA 3175 CNRS at CINVESTAV-IPN.

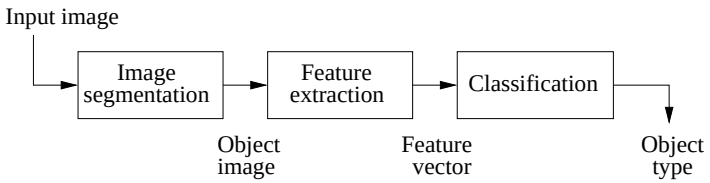

Fig. 1. The three phases of a pattern recognition system.

Feature selection deals with extracting features for differentiating one class of objects from another. The output of this stage is a vector of values of the measured features.

The last stage is called **classification**, or recognition. This is the process that assigns a label to each object based on the information provided by their descriptors.

As will be seen in this chapter, Evolutionary Algorithms (EAs) have been applied to all three stages of the pattern recognition process [3]. In fact, some authors such as Rizki et al. [4] have applied EAs to each stage of a pattern recognition system, with minimum human intervention, with the aim of obtaining the best possible (overall) pattern recognition system.

From the several metaheuristics that currently exist (see for example [5, 6]) for solving optimization and classification problems, EAs, which emulate the natural selection mechanism, have become one of the most popular, mainly because of their ease of implementation, their flexibility and their high effectivity in a wide variety of tasks [7, 8, 9, 10]. Thus, this chapter aims to provide a general (although not comprehensive due to obvious space limitations) overview on the use of EAs for solving pattern recognition tasks.

The remainder of this chapter is organized as follows. Section 2 presents a short introduction to evolutionary algorithms, including two examples that illustrate their use. Within these two examples, two specific evolutionary algorithms (namely, genetic algorithms and differential evolution) are discussed in more detail. Then, in Section 3 we provide a brief review of several applications representative of the work done regarding the use of EAs in any of the three previously indicated stages of a pattern recognition task. Some possible paths for future research regarding the use of EAs in pattern recognition are briefly discussed in Section 4. Finally, the main conclusions of this chapter are drawn in Section 5.

2 Basic Notions of Evolutionary Algorithms

Evolutionary algorithms are bio-inspired metaheuristics that attempt to emulate Charles Darwin's natural selection mechanism (i.e., the "survival of the fittest" principle) with the purpose of solving (mainly optimization) problems [8]. Metaheuristics are high-level frameworks that combine basic (low-level)

heuristic methods to explore the search space of a problem in a more efficient and effective way [11]. A *heuristic* is a technique that searches for good solutions at a reasonable computational cost, but without guaranteeing optimality. In fact, in some cases, heuristics cannot even determine how far is a certain solution from the optimum [12].

The idea of getting inspiration from the natural selection mechanism to solve problems is not new, since it can be traced back to the 1930s [13]. Nevertheless, it was until the 1960s that these early ideas were actually implemented. Three are the main paradigms considered within EAs: genetic algorithms [14, 15, 7], evolution strategies [16, 17, 18] and evolutionary programming [19, 20]. Each of them was developed independent from the others, and with different motivations (e.g., genetic algorithms were originally developed to solve machine learning problems, whereas evolution strategies were originally developed to solve optimization problems). There have been also variations of some of these approaches. The most remarkable is genetic programming [21, 22, 23, 24], which is a variation of the genetic algorithm that uses tree-based encoding, instead of the original fixed-length binary strings adopted in genetic algorithms.

EAs, in their different versions and variations have been found to be very effective for solving a wide variety of optimization and classification problems [7, 18, 20]. The main reasons for their popularity are their generality (they require little domain-specific information), ease of implementation and use, combined with their high effectiveness in solving highly complex problems [7].

The basic operation of an EA can be summarized as follows. First, they generate a set of possible solutions (called a "population") to the problem that is being solved. Such a population is normally generated in a random manner. Each solution in the population (called an "individual") encodes all the decision variables of the problem. In order to assess the suitability of such individuals, a fitness function must be defined. Such a fitness function is a variation of the objective function of the problem that we wish to solve and is used as a relative measure of performance among individuals (i.e., solutions that represent better objective function values are the fittest). Then, a selection mechanism must be applied in order to decide which individuals will "mate." This selection process is normally based on the fitness contribution of each individual (i.e., the fittest individuals have a higher probability of being selected). Upon mating, a set of children or "offspring" are generated. Such offspring are "mutated" (this operator produces a small random change, with a low probability, on the contents of an individual), and become the population to be evaluated at the following iteration (called a "generation"). This process is repeated until reaching a stopping condition (normally, a maximum number of generations).

The previous description corresponds to a general EA, but each specific EA adopts variations of this procedure. For example, a genetic algorithm will perform crossover (several possible crossover schemes exist), whereas evolutionary programming uses only mutation. It is also worth indicating that there

are a few recent metaheuristics that are also normally considered as EAs. From them, the two most popular are differential evolution (DE) [25] and particle swarm optimization (PSO) [26].

Next, we will illustrate two application examples of the use of EAs in pattern recognition. In these examples, two specific types of EAs will be briefly described (namely, genetic algorithms and differential evolution).

2.1 Example 1: Clustering

Clustering can be seen as a previous stage to classification, in which groups of objects are labeled with an integer value and assigned into subsets (the so-called *clusters*), so that objects in the same cluster are similar according to some measure. Clustering is considered a method of unsupervised learning, and is commonly used for statistical data analysis in many disciplines, including data mining, pattern recognition, image analysis, machine learning and bioinformatics [27, 28].

Perhaps the most popular clustering algorithm in current use is k-means [29]. The k-means algorithm produces a partition of n observations ($\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$), where $\mathbf{x}_i$ is a vector of features of dimension d, into k sets ($k < n$) $\mathbf{S} = \{S_1, S_2, \ldots, S_k\}$ such that the distances form each observation to the nearest partition is minimized:

$$\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_i - \boldsymbol{\mu}_i\|^2, \tag{1}$$

where $\boldsymbol{\mu}_i$ is the mean of the observations in the set i.

Very recently, it was proved that the complexity of the problem tackled by the k-means algorithm is NP-hard [30] in a general Euclidean space of dimension d, even for 2 clusters. It is also NP-hard even in 2 dimensions [31] for a general number of clusters. NP-hard complexity of a problem implies that, at least up to now, there is no procedure that can solve it in a polynomial time. In other words, this is really a very difficult task, which therefore justifies the use of a metaheuristic to solve it. Here, we will use a genetic algorithm to solve this problem.

A Simple Genetic Algorithm

Genetic algorithms (GAs) 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 GA is illustrated in Figure 2.

First, an initial population is randomly generated, as indicated in the description of a general EA. It is worth indicating, however, that deterministic

```
Generate initial population of size M in a random manner;
Repeat
    Evaluate fitness of each individual within M;
    Select (normally in a probabilistic way) individuals
        (called parents) based on fitness;
    With probability P_c, apply the crossover operator
        between pairs of parents.
    Apply the mutation operator to all the children
        generated from the previous step with a probability P_m.
    Apply elitism (i.e., the best individual in the population
        remains intact for the following generation).
Until the stop condition is reached
```

Fig. 2. Pseudocode of a simple genetic algorithm.

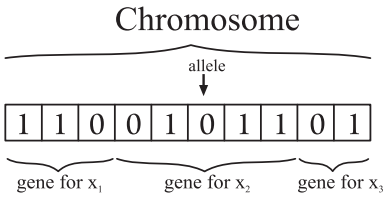

Fig. 3. Example of the binary encoding traditionally adopted with the genetic algorithm.

or semi-deterministic procedures can also be used for generating the initial population (using, for example, a greedy procedure).

The individuals of the population of a GA will be a set of strings of characters (letters and/or numbers) called *chromosomes* that represent all the possible solutions to the problem. Chromosomes are made of *genes* that correspond to each of the decision variables of the problem. Finally, genes are made of *alleles* which correspond to characters or symbols in the alphabet used as a basis for the encoding. This notation is graphically depicted in Fig. 3.

One aspect that has great importance in the case of the genetic algorithm is the encoding of solutions, since GAs normally use an indirect representation of the solutions of the problem (unlike evolution strategies that typically adopt a direct representation of solutions, by operating over vectors of real numbers). Traditionally, GAs adopt a binary encoding regardless of the type of decision variables of the problem to be solved, mainly because of the fact that this sort of encoding can be considered as universal [7], but biological arguments have also been provided to favor this sort of encoding [32]. Nevertheless, other types of encodings are also possible and have been used with genetic algorithms (see for example [33, 34, 35]).

After defining the encoding to be adopted by the GA, a *fitness function* value must be computed for each of the chromosomes in the population. These

fitness values measure the quality of the solution encoded by the chromosome and represent a relative performance measure that will allow us to know which solutions are preferable over others. Knowing each chromosome's fitness, a *selection* process takes place to choose the individuals (presumably, the fittest) that will become the parents of the following generation. A variety of selection schemes exist [36], including *roulette wheel* selection [37], *stochastic remainder* selection [38, 39], *stochastic universal* selection [40, 41], *ranking* selection [42] and *tournament* selection.

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. This operator is applied with a certain probability P_c to pairs of individuals selected to be parents (P_c is normally set between 60% and 100%). When using binary encoding, there are three main ways of performing crossover:

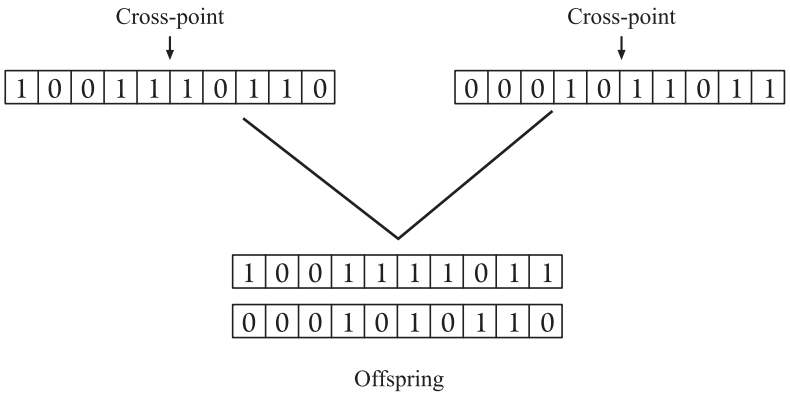

Fig. 4. Use of a single-point crossover between two chromosomes. Notice that each pair of chromosomes produces two descendants for the next generation.

1. *Single-point crossover*: A position of the chromosome is randomly selected as the crossover point as indicated in Fig. 4.
2. *Two-point crossover*: Two positions of the chromosome are randomly selected for exchanging chromosomic material, as indicated in Fig. 5.
3. *Uniform crossover*: This operator was proposed by Syswerda [43] and can be seen as a generalization of the two previous crossover techniques. In this case, for each bit in the first offspring it decides (with some probability P_c) which parent will contribute its value in that position. The second offspring will receive the bit from the other parent. Although for some problems uniform crossover presents several advantages over other crossover techniques [43], in general, one-point crossover seems to be a bad choice, but there is no clear winner between two-point and uniform crossover [44, 34].

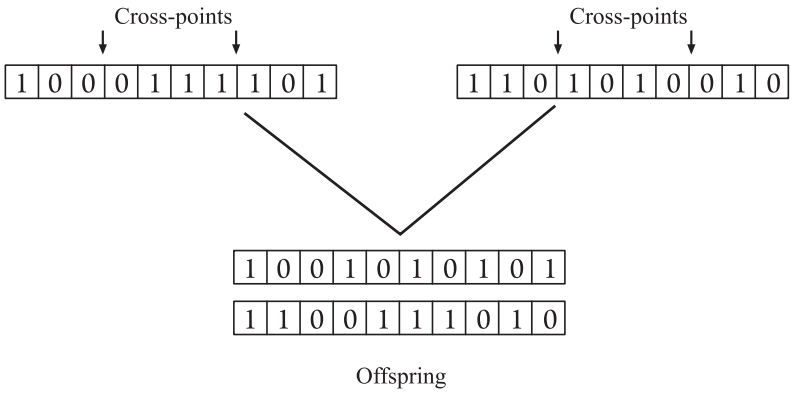

Fig. 5. Use of a two-point crossover between two chromosomes. In this case the genes at the extremes are kept, and those in the middle part are exchanged.

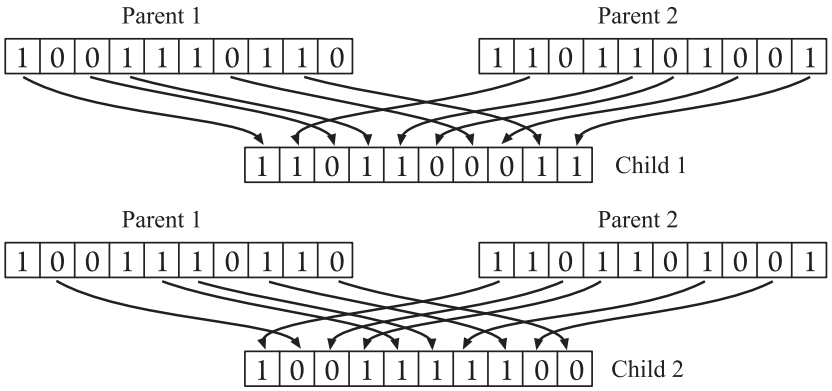

Fig. 6. Use of 0.5-uniform crossover (i.e., adopting a 50% probability of crossover) between two chromosomes. Notice how half of the genes of each parent go to each of the two children. First, the bits to be copied from each parent are selected randomly using the probability desired, and after the first child is generated, the same values are used to generate the second child, but inverting the source of procedence of the genes.

Other crossover operators are, of course, possible, mainly when adopting other encodings (see for example [45]).

The offspring generated by the crossover operator are subject to *mutation*, which is a 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. This operator is applied with a probability P_m to each allele in a chromosome (P_m normally adopts a low probability that goes from 1% up to 10% as a maximum). The use of this operator allows the introduction of new chromo-

somic material to the population and, from a theoretical perspective, it assures that—given any population—the entire search space is connected [46].

Finally, the individual with the highest fitness in the population is retained, and it passes intact to the following generation (i.e., it is not subject to either crossover or mutation). This operator is called *elitism* and its use is required to guarantee convergence of a simple GA, under certain assumptions (see [47] for details).

The previous procedure must be repeated a certain number of times, until reaching a stopping criterion. The most commonly adopted stopping criterion is to adopt a (pre-defined) maximum number of iterations (or *generations*), but other criteria are also possible [7].

Although GAs can benefit from self-adaptation mechanisms that allow their parameters to be defined in an automated way [48], their users normally fine-tune their parameters (population size, crossover and mutation rates, etc.) by hand, using a trial-and-error process.

The Example

To show how this simple GA works, we will use the k-means problem. Let's assume that we have five observations and three clusters. Thus, an individual can be represented as $(1, 2, 1, 3, 2)$ meaning that the first and third observations form the first cluster. Analogously, the second and fifth observations form the second cluster, and the third cluster is formed with only the fourth observation.

The binary encoding corresponding to the above individual would be the following: $([0, 1], [1, 0], [0, 1], [1, 1], [1, 0])$. Any of the previously discussed crossover operators can be used in this case. Mutation is also applied as indicated before (using a NOT operator). The fitness value of each individual is obtained using Eq. (1). In this case, and considering the encoding adopted by the GA, the means must be computed from the code sets, each time we need to obtain the fitness value of an individual.

The use of a population, combined with the application of mutation, reduces the probability that the GA gets trapped in local minima. When using a GA, the use of a randomly generated population (using a uniform distribution) to start the search, solves the initialization of the k-means algorithm, which is known to have a significant impact on its performance [49].

It is worth indicating that GA-based clustering algorithms [50, 51] have shown superior performance than traditional clustering algorithms [49].

2.2 Example 2: Robust Ellipse Fitting

A silhouette of an ellipse, or a data set with elliptical form, can be detected by fitting. Given a function D that calculates the distance from an ellipse $\mathbf{x}$ to a point $\mathbf{p}$, the best ellipse is that which minimizes the function:

$$g : \mathbb{R}^5 \rightarrow \mathbb{R},$$
$$g = \sum_{i=1}^{n} D^2(\mathbf{x}, \mathbf{p}_i), \tag{2}$$

where $\mathbf{p}_i = [x_i, y_i]^{\mathrm{T}}$ is the vector representing the coordinates of each point i, $i = 1, 2 \ldots n$, and the ellipse is represented by a vector of five parameters $\mathbf{x} = [a, b, x_c, y_c, \alpha]^{\mathrm{T}}$, where a and b are the semimajor and semiminor axes of the ellipse in a canonical form (centered at the origin of the coordinate system), (x_c, y_c) are the coordinates of the new origin for the translated ellipse, and α is the rotation angle between the semimajor axis and the original x axis.

To solve the problem stated in Eq. (2) is relatively simple by using the least squares method. The least squares procedure is based on the observation that the minimum of Eq. (2) is located at the place where its derivative (or its gradient) is equal to zero. Using the algebraic distance, the problem is linear and generates the most efficient algorithm to solve it [52]. Considering the real orthogonal distance between each point and the ellipse, this becomes a nonlinear problem which can be solved using the Gauss-Newton method [53].

The problem here is that using the algebraic distance generates distorted ellipses if points are not provided with sufficient accuracy. Also, the least squares method is not resistant to the presence of outliers in the data: points far away from the ellipse will be considered using the square of their distances. Furthermore, in the case of the nonlinear problem, an initial point, located very near to the actual optimum is required in order to solve it using mathematical programming techniques.

In order to obtain better results, we will use the linear solution to the problem as a starting point for the nonlinear algorithm. It is worth noting, however, that by doing this, the outliers affect even more to the linear solution.

Taking the sum of the distances, instead of their squared values, generates a new problem that will not present the same difficulties as the least squares procedures previously described. In this case, the problem consists of minimizing the following function:

$$g_1 : \mathbb{R}^5 \rightarrow \mathbb{R},$$
$$g_1 = \sum_{i=1}^{n} D(\mathbf{x}, \mathbf{p}_i), \tag{3}$$

The minimization of Eq. (3) is robust to the presence of outliers because every distance will be taken as it is, rather than adopting its squared value. The problem now is that Eq. (3) cannot be solved using any conventional least squares approach. Thus, the use of a metaheuristic is now appropriate. Something interesting here is that using an EA to solve this problem is much simpler [54, 55] than adopting a nonlinear least squares procedure, because we only need a procedure that computes the value of the function g_1 (we do not need to compute gradients or to perform any matrix inversion).

In this case, it is not possible to use the simple GA described in Section 2.1, because the most suitable representation for the solutions of this problem are vectors of real numbers. Although it is possible to use GAs with real-numbers encoding (see for example [56, 57]), we will adopt in this case an EA that has been found to be very powerful when dealing with this sort of problems (i.e., those in which the decision variables are real numbers): differential evolution [58, 25].

Differential Evolution

Differential Evolution was proposed by Kenneth Price and Rainer Storn in the mid 1990s [59, 60, 25]. DE is an evolutionary (direct-search) algorithm which has been mainly used to solve continuous optimization problems. DE shares similarities with traditional EAs. Unlike simple GAs [7], DE does not adopt binary encoding. Also, it does not use a probability density function to self-adapt its parameters as done with Evolution Strategies [61]. Instead, DE performs mutation based on the distribution of the solutions in the current population. Thus, the search directions and any possible step sizes depend on the location of the individuals that were selected to calculate the mutation values.

The most popular nomenclature adopted to refer to the different DE variants is called *"DE/rand/1/bin"*, where "DE" means Differential Evolution, the word "rand" indicates that individuals selected to compute the mutation values are chosen at random, "1" is the number of pairs of solutions chosen and finally "bin" means that a binomial recombination is used. The corresponding algorithm of this variant (*"DE/rand/1/bin"*) is shown in Fig. 7 and is the most popular in the specialized literature.

The "CR" parameter controls the influence of the parent in the generation of the offspring. Higher values mean less influence of the parent. The "F" parameter scales the influence of the pairs of solutions that are selected to obtain the mutation value (only one pair in the case of the algorithm in Fig. 7). The stopping criterion of DE is normally a maximum number of iterations (as in the case of the simple GA), but other, more elaborate criteria are also possible (see for example [62]).

Several DE variants are possible. To exemplify this point, we took from the paper by Mezura et al. [63] the eight DE variants adopted, each of which will be briefly described next. The modifications from variant to variant are in the recombination operator used (steps 9 to 15 in Fig. 7) and also in the way individuals are selected to calculate the mutation vector (step 7 in Fig. 7). The variants adopted by Mezura et al. [63] are the following:

- Four variants whose recombination operator is discrete, always using two individuals: the original parent and the DE mutation vector (step 11 in Fig. 7). Two discrete recombination operators: binomial and exponential. The main difference between them is that for binomial recombination, each

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

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

variable value of the offspring is taken at each time from one of the two parents, based on the value of "CR". On the other hand, in the exponential recombination, the value of each variable that forms the offspring is taken from the first parent until a random number surpasses the "CR" value. From this point, all the remaining offspring variable values will be taken from the second parent. These variants are called: *"DE/rand/1/bin"*, *"DE/rand/1/exp"*, *"DE/best/1/bin"* and *"DE/best/1/exp"* [64]. The "rand" variants select at random to all the individuals to compute mutation and the "best" variants use the best solution in the population besides the random ones.

- Two variants with arithmetic recombination, which, unlike discrete recombination, are rotation invariant. These are *"DE/current-to-rand/1"* and *"DE/current-to-best/1"* [64]. The only difference between them is that the

first selects the individuals for mutation at random and the second one uses the best solution in the population besides random solutions.

- *"DE/rand/2/dir"* [65], which incorporates objective function information to the mutation and recombination operators. The aim of this approach is to guide the search to promising areas faster than traditional DE. Their authors argue that the best results are obtained when the number of pairs of solutions is two [65].
- Finally, a variant with a combined discrete-arithmetic recombination, the *"DE/current-to-rand/1/bin"* [64].

Each variant's implementation details are summarized in Table 1.

The Example

In our example, the vectors of real numbers for each individual will be of size 5, since we have five decision variables (a, b, x_c, y_c, α). The population is randomly initialized within the allowable range of the decision variables. In our example, a and b can vary between 1 and one half of the size of the input image (if we consider that every point also represents a pixel position in an image), (x_c, y_c) can be inside the image, and $\alpha \in [0 : \pi]$.

For the fitness value, we use in this case the value of the function g_1 indicated in Eq. (3). The results obtained by DE are shown in Fig. 8, in which they are compared with respect to those obtained with the linear algorithm. It can be clearly seen that the ellipse fitted by DE is much better than the other one.

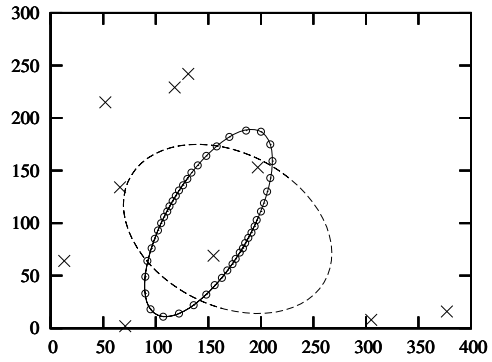

Fig. 8. 45 ellipse points and 10 outliers. We show with a solid line to the ellipse fitted by differential evolution, using the sum of distances. With a dashed line we show the ellipse fitted with the linear algorithm, according to the sum of the squared distances. Clearly the last one is very distortioned and is not resistant to the presence of outliers.

Nomenclature	Variant
rand/p/bin	$u_{i,j} = \begin{cases} x_{r_3,j} + F \cdot \sum_{k=1}^{p} (x_{r_1^p,j} - x_{r_2^p,j}) & \text{if } U_j(0,1) < CR \text{ or } j = j_r \\ x_{i,j} & \text{otherwise} \end{cases}$
rand/p/exp	$u_{i,j} = \begin{cases} x_{r_3,j} + F \cdot \sum_{k=1}^{p} (x_{r_1^p,j} - x_{r_2^p,j}) & \text{from } U_j(0,1) < CR \text{ or } j = j_r \\ x_{i,j} & \text{otherwise} \end{cases}$
best/p/bin	$u_{i,j} = \begin{cases} x_{best,j} + F \cdot \sum_{k=1}^{p} (x_{r_1^p,j} - x_{r_2^p,j}) & \text{if } U_j(0,1) < CR \text{ or } j = j_r \\ x_{i,j} & \text{otherwise} \end{cases}$
best/p/exp	$u_{i,j} = \begin{cases} x_{best,j} + F \cdot \sum_{k=1}^{p} (x_{r_1^p,j} - x_{r_2^p,j}) & \text{from } U_j(0,1) < CR \text{ or } j = j_r \\ x_{i,j} & \text{otherwise} \end{cases}$
current-to-rand/p	$\mathbf{u}_i = \mathbf{x}_i + K \cdot (\mathbf{x}_{r_3} - \mathbf{x}_i) + F \cdot \sum_{k=1}^{p} (\mathbf{x}_{r_1^p} - \mathbf{x}_{r_2^p})$
current-to-best/p	$\mathbf{u}_i = \mathbf{x}_i + K \cdot (\mathbf{x}_{best} - \mathbf{x}_i) + F \cdot \sum_{k=1}^{p} (\mathbf{x}_{r_1^p} - \mathbf{x}_{r_2^p})$
current-to-rand/p/bin	$u_{i,j} = \begin{cases} \mathbf{x}_{i,j} + K \cdot (\mathbf{x}_{r_3,j} - \mathbf{x}_{i,j}) + F \cdot \sum_{k=1}^{p} (\mathbf{x}_{r_1^p,j} - \mathbf{x}_{r_2^p,j}) & \text{if } U_j(0,1) < CR \text{ or } j = j_r \\ x_{i,j} & \text{otherwise} \end{cases}$
rand/2/dir	$\mathbf{v}_i = \mathbf{v}_1 + \frac{F}{2}(\mathbf{v}_1 - \mathbf{v}_2 + \mathbf{v}_3 - \mathbf{v}_4)$ where $f(\mathbf{v}_1) < f(\mathbf{v}_2)$ and $f(\mathbf{v}_3) < f(\mathbf{v}_4)$

Table 1. DE variants adopted by Mezura et al. [63]. j_r is a random integer number generated within the range $[0, n]$, where n is the number of decision variables of the problem. $U_j(0,1)$ is a real number generated at random between 0 an 1. Both numbers are generated using a uniform distribution. In their experiments, Mezura et al. [63] used $p = 1$.

3 A Review of EAs in Pattern Recognition

In this section, we will briefly review some representative work on the use of EAs for pattern recognition tasks. Our review will provide examples of all three stages of the pattern recognition process: (1) segmentation, (2) feature selection and (3) classification. Each of these groups of applications are discussed in the following subsections.

3.1 Segmentation

Image segmentation denotes a process by which a raw input image is partitioned into non-overlapping regions such that each region is homogeneous and connected. A segmented image is often considered connected if there exists a connected path between any two pixels within the region. A region is considered homogeneous if all of its pixels satisfy a homogeneity criterion defined over one or more pixel attributes such as intensity, texture, color, range, etc. Computing such an image partition is a problem of very high combinatorial complexity. Given the astronomical size of the search space, an exhaustive or near-exhaustive enumeration of all possible image partitions to arrive at a segmented image is usually infeasible. This motivates the use of EAs in this problem.

In [66] Bhandarkar and Zhang use three hybrid EAs for the segmentation of gray level images: (1) a GA hybridized with simulated annealing (SA) [67] called SA-GA, (2) a GA hybridized with microcanonical annealing (MCA) [68] called MCA-GA, and (3) a GA hybridized with a random cost algorithm (RCA) [69] called RCA-GA. It should be noted that SA, MCA and RCA are all stochastic hill-climbing search techniques (i.e., they are local search engines). The problem was defined as one of minimization, in which the fitness function incorporated both region contour (or edge information) and region gray-scale uniformity (such uniformity was quantified using gray-level variance values). Clearly, the desired segmentation corresponded to the global minimum of the cost function proposed by the authors. The authors noted, however, that their fitness function presented many local minima. The results of the three hybrid schemes were compared with respect to those of a simple GA using several images, both with and without (Gaussian) noise. RCA-GA was the fastest approach, closely followed by MCA-GA. However, it was SA-GA (the slowest among the hybrid approaches) the one that obtained the best overall results in terms of both visual quality and cost value of the final segmentation. All the hybrid approaches had a better performance than the simple GA, which clearly showed the usefulness of local search in this case.[2]

Jiang and Yang proposed in [71] a hybrid approach that combines features of genetic algorithms [7] and tabu search [72]. The proposed approach, which

[2] Population-based approaches (e.g., EAs) hybridized with a local search procedure are also known as **memetic algorithms** [70].

was called "evolutionary tabu search" (ETS) was used for the segmentation of cell images. Knowing that most cells in the human body have ellipse-like boundaries, the authors used an ellipse equation to describe the boundary of a cell. Since the definition of an ellipse requires five parameters (two to determine its location and three more to determine its size and orientation), the authors used these five values as the decision variables to be obtained. The goal was, evidently, to fit the cell boundary as well as possible, and, therefore, the fitness function was defined in such a way that it counted the number of points that were within a certain (fixed) distance from the ellipse. In order to find the edge points, the authors adopted the Canny operator [73]. The proposed hybrid was compared with respect to its two components considered separately (i.e., the genetic algorithm and tabu search), showing to be superior to both of them. The authors reported that their proposed approach was able to find consistently (i.e., there was little variation of results over several independent runs) near-optimal results for several images with red blood cells.

Bocchi et al. presented in [74] an EA for image segmentation. The main idea of this approach is to perform a colonization of a bidimensional world (i.e., the image) by a certain number of populations, each of which represents a different region of the image. The individuals of each population, competed in order to occupy all the available space and to adapt to the specific local environmental characteristics of the world. This approach was inspired on the famous game of "Life" [75]. The authors validated their proposed approach using several synthetic images. Their results produced significant improvements with respect to those found by the well-known fuzzy c-means clustering algorithm [76]. The authors indicated that their approach can be used for the segmentation of gray-scale, color and textural images. Indeed, they indicated that their approach can be extended to any vector-valued parametric images, regardless of their number of components.

Krawiec et al. provided in [77] a review of work on the use of genetic programming (GP) [22] for object detection and image analysis. As indicated before, GP refers to a variation of the genetic algorithm in which a tree-encoding is adopted. This special kind of encoding evidently requires of different alphabets and specialized operators, since we are, in fact, evolving programs rather than simple vectors of decision variables. The trees used in GP consist of both functions and terminals. The most commonly adopted functions are the following [22]:

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

Terminals are typically variables or constants, and can be seen as functions that take no arguments. GP incorporates fitness-based selection, crossover and mutation (obviously, all of them are modified so that they can properly deal with trees), but also adds special procedures for generating the initial population (e.g., a maximum tree height is normally enforced to avoid an excessive memory use), as well as other operators for a variety of tasks (e.g., to destroy a certain percentage of the population in order to improve diversity, or to "encapsulate" or protect certain subtrees that we want to keep). Because of its nature, GP provides a powerful tool for building scalable and adaptive image analysis systems, which use raw image data as input and produce complete recognition systems as their output. Krawiec et al. [77] point out, however, that the main drawback of GP is that (as any other EA) it normally requires a considerable number of fitness function evaluations, and each of these evaluations are quite expensive (computational speaking) in this case. They also indicate that most of the applications that they reviewed are focused on feature-based recognition, in which the evolved system is able to discriminate between positive and negative examples based on certain features of the image. However, they found very few applications focused on model-based recognition in which the underlying assumption is that a database of models of recognized objects is available and comparisons are then done between the input image and the object models. This sort of approach allows for the recognition of more complex (even compound) objects, but has been rarely adopted with GP.

3.2 Feature Selection

Feature selection refers to the selection of the most relevant features (e.g., of a class of objects) so that we can build robust learning models (e.g., for classifying classes of objects). By robustness we refer to finding the minimum subset of features that are useful to keep in order to obtain an efficient and improved solution to the problem of our interest (e.g., to classify a set of objects). This task is important, since normally not all the available features are useful and keeping any unnecessary features increases the computational cost required to solve the problem. Performing an optimal feature selection requires an exhaustive search of all possible subsets of features. This makes this problem suitable for using EAs, because of the very large search space normally involved.

In [78], Muni et al. presented an online feature selection algorithm based on GP. In fact, the use of the tree-based encoding of GP also allows for the design of the classifiers, which is something they do for a multicategory classification problem. The authors generated the initial population in such a way that the initialization process itself, generated classifiers that had a high probability of using smaller feature subsets. The fitness function adopted by the authors assigns a higher fitness value to any classifier that is able to classify more samples using less features. Indeed, this can be seen as a multiobjective

fitness function [79], since it performs both feature selection and the design of classifiers at the same time. The authors also proposed two crossover operators specially designed to perform feature selection. The authors compared results with respect to those reported in the specialized literature for several data sets that go from low (four) to high (over 7000) dimensionality. Their proposed approach provided better performance for both two-class and multiclass problems, even in problems having redundant or bad features (which were artificially added). The main limitation of this approach is only its applicability, which is constrained to numerical attributes only, because their GP implementation adopted arithmetic functions to design the classifiers.

Watchareeruetai et al. [80] adopted a variation of Linear Genetic Programming (LGP)[3] [81] for extracting features from images. The authors indicated that the main source of problems when using LGP for image feature extraction is the excessive redundancy associated with the encoding adopted by this technique. Such redundancy can substantially increase the computational cost of LGP, since the same costly solutions are being evaluated more than once. In order to deal with this problem, the authors proposed a transformation of the LGP representation into a canonical form that has no redundancies. The experiments conducted by the authors indicated reductions in computational time that go from 7% up to 62% with respect to the use of the original LGP encoding.

Kowaliw et al. [82] adopted cartesian GP[4] [84] to define a set of transforms on the space of grayscale images which were meant to facilitate a further classification process. The idea is that these transforms (which are really programs) emphasize distinguishing characteristics. The authors applied their proposed approach for detecting muscular dystrophy-indicating inclusions in cell images. In their experiments, the authors were able to discover a set of features that could achieve 91% recognition of healthy cells, and an 88% recognition of sick cells. These accuracies provided a 38% improvement over predefined features alone. However, the authors considered as more important the fact that this approach may constitute an important step towards having a GP-based recognition system that automatically adapts to a given database without any human intervention.

Guo et al. [85] proposed an automatic image pattern recognition system which was used for classification of medical images. This system adopts (apparently, for the first time, as claimed by the authors) a generalized primitive texture feature extraction technique based on a histogram region of interest by thresholds (HROIT), which is used to characterize the Oculopharyngeal Muscular Dystrophy (OPMD) disease. They also proposed a new technique, based

[3] LGP is a GP [22] variant that evolves sequences of instructions from an imperative programming language. The term *linear* is used in this case to denote the structure of the imperative program representation [81].

[4] Cartesian Genetic Programming was proposed in [83] with the purpose of evolving digital circuits and represents programs as directed graphs.

on the integration of genetic programming (GP) and the expectation maximization (EM) algorithm [86] (called GP-EM), for generating feature functions automatically, based on the primitive features obtained from HROIT. The GP-EM system makes simpler the learning task by generating a single feature (which is actually a program generated by GP). The authors showed that this approach led to higher classification accuracies (90%, on average, in diagnosing the OPMD disease) and lower standard deviations, being able to outperform another classification system based on a Support Vector Machine (SVM) [87].

Raymer et al. [88] proposed an approach in which a GA was used to perform, simultaneously, feature selection, feature extraction and classifier training. The GA was used to reduce the dimensionality of the feature set. Basically, the GA transformed a set of patterns into a lower dimensionality (a set of weight vectors that scale the individual features of the original pattern vectors was used for this sake). The aim was to reduce the dimensionality as much as possible, while maximizing the classification accuracy. The authors also adopted a binary masking vector to perform a selection of a subset of the features under consideration. This was done while performing the dimensionality reduction, since the GA used both vectors (the weight vectors and the binary masking vector) in its chromosomes. Each of the resulting subsets of features were evaluated in terms of their classification accuracy on some test data using a nearest neighbor classifier. They also used this approach in combination with the k nearest neighbor classification rule to allow linear feature extraction (instead of binary vectors, real numbers were used in this case). The proposed approach was tested on medical and on biochemical data, producing very competitive results in all cases, while using less features than the other classifiers with respect to which it was compared.

3.3 Classification and Clustering

As indicated before, clustering can be seen as a previous stage to classification, and even as a simpler form of classification, which makes it very difficult to distinguish one task from the other. Because of that, both of them are considered in this section.

Iglesia et al. [89, 90] used a multi-objective GA called *Nondominated Sorting Genetic Algorithm-II* (NSGA-II) [91] for doing partial classification (this is called the *nugget discovery task*). The aim was to maximize both confidence and coverage of the rules. The initial population of the GA was not randomly generated. Instead, the authors used a special procedure that looked at the data to ensure that no rules with zero coverage were produced (as happens when a randomly generated population is adopted). In order to achieve this, a default rule was adopted (in this default rule, all limits were maximally spaced and all labels were included), and the remainder of the population were just mutations of the default rule. The authors compared results with respect to another approach which was developed by some of the same authors. For their

validation, the authors used data sets taken from the UCI repository [92]. The proposed approach was able to produce sets of rules of similar quality as the other approach, and was even able to outperform it in some cases.

EAs have been widely used for clustering, as made evident in the survey on GA-based clustering written by Sheikh et al. [93]. In this review, the authors indicated that one of the main advantages of EAs, when used for clustering, is that, unlike classical clustering techniques (e.g., k-means [29], fuzzy c-means [76], etc.), they do not require the number of clusters as an input parameter, since this value can be produced during the search. This review also showed a rich variety of applications, including image compression, microarray data analysis, document clustering and text clustering, among others. Finally, the authors indicated that in their review they found that GAs had only being applied to distance-based clustering algorithms (e.g., k-means) but not with other types of clustering algorithms.

Bandyopadhyay and Maulik presented in [94] a GA for the automated clustering of data sets. The proposed approach can evolve the number of clusters while performing the clustering of the data. The authors adopted a special encoding that contains both real numbers (which encode the centers of the clusters) and "don't" care symbols (which are used to encode a variable number of clusters). The fitness of individuals was computed using the Davies-Bouldin index [95]. The proposed approach was validated using four artificial and two real-world data sets in which the number of clusters goes from two to six, and the number of dimensions goes from two to nine. In their experiments, the authors considered both overlapping and non-overlapping data sets. In all cases, the results were found to be quite competitive.

In a further paper, Bandyopadhyay et al. [96] proposed the constrained elitist multiobjective genetic algorithm based classifier (CEMOGA-Classifier) for developing nonparametric classifiers. The proposed approach can overcome problems common to traditional (single-objective) classifiers, such as overfitting/overlearning and ignoring smaller classes. In the proposed classifier, the authors considered three objectives: minimize (1) the number of misclassified training points and (2) the number of hyperplanes, and maximize (3) the product classwise correct recognition rates. Since CEMOGA-Classifier generates several solutions (called *nondominated* [79]), and it is desirable to have only one (which corresponds to the desired classifier), the authors adopted a validity function to select only one solution from the nondominated set. This validity function is really an aggregating function that combines the values achieved by the classifier in each of the three objectives considered during the optimization process. This validity function allows the user to weight the importance that he/she wishes to give to each of these objectives. Results were compared with respect to those produced by two well-established multiobjective evolutionary algorithms: NSGA-II [91] and the Pareto Archived Evolution Strategy (PAES) [97]. In general, CEMOGA-Classifier was able to approximate the class boundaries of the data sets adopted using a smaller number of hyperplanes, which indicates a superior generalization capability.

4 Future Research Directions

As has been shown in the previous sections, the use of EAs in pattern recognition is a very active research area. There are, however, several other topics within this area that still represent interesting research topics. The following are a few examples:

- **Integration:** One of the long-term goals of using EAs in pattern recognition must be the development of fully automated systems that can be applied to different databases with minimum (or no) human intervention. This may require to combine EAs with other approaches (e.g., fuzzy logic and/or machine learning techniques) as well as the design of new architectures that allow an efficient and effective integration of different types of approaches throughout the different stages involved in a pattern recognition process. The use of multiobjective optimization techniques (which are designed to solve problems in which we aim to optimize two or more (normally conflicting) objectives) may be useful for this task [79] and have, indeed, been used (and have been raising increasing interest) in a variety of pattern recognition tasks (see for example [98, 99, 100]).

- **Efficiency:** As indicated before, a critical aspect that could certainly limit the applicability of EAs in pattern recognition is their computational cost. Although EAs by themselves are computationally inexpensive, they normally require to evaluate the fitness of several hundreds or thousands of solutions in order to obtain the necessary information to guide the search. In many pattern recognition tasks (e.g., in segmentation) these fitness evaluations are computationally expensive. There has been little work until now regarding the incorporation of fitness approximation techniques [101] to EAs adopted for pattern recognition tasks. Such techniques could provide an interesting alternative to improve the efficiency of an EA, at the expense of sacrificing some accuracy, if this is affordable at the application at hand.

- **Use of other Metaheuristics:** In recent years, other bio-inspired metaheuristics have become increasingly popular in a wide variety of applications [102]. These metaheuristics also have a lot of potential in pattern recognition tasks, but their use in this domain still remains relatively scarce. Representative examples of these new metaheuristics are the following:
  - **Particle Swarm Optimization**: This is actually another type of EA which was originally proposed by James Kennedy and Russell C. Eberhart in the mid-1990s [103, 26]. This metaheuristic simulates the movements of a flock of birds which aim to find food. In this approach, the behavior of each individual (called *particle*) is affected by either the

best local (i.e., within a certain neighborhood) or the best global individual. As other EAs, it also uses a population as well as a fitness measure. However, because of its design, this metaheuristic allows individuals to benefit from their past experiences, which is a feature that is normally inexistent in traditional EAs. This technique has been widely applied in a variety of problems [26].

- **Artificial Immune Systems**: Our immune system can be seen as a highly parallel intelligent system that is able to learn and retrieve previous knowledge (in other words, it has "memory"), while solving complex recognition and classification tasks (namely, detecting antigens that invade our body). These features make immune systems quite attractive from a computational point of view, and has motivated a lot of research on the development of mathematical and computational models that emulate its operation. Artificial immune systems were introduced in the mid-1990s and since then, have attracted increasing interest from researchers who have used it for a variety of tasks, including a few classification and pattern recognition problems [104, 105, 106].
- **Ant Colony Optimization**: This is a metaheuristic inspired by the behavior shown by colonies of real ants which deposit a chemical substance on the ground called *pheromone* [107, 108]. The pheromone influences the behavior of the ants: they tend to take those paths in which there is a larger amount of pheromone. Pheromone trails can thus be seen as an indirect communication mechanism used by the ants. This system also presents several interesting features from a computational perspective, and has triggered a significant amount of research. The first of these ant-based systems, called *ant system* was originally proposed for the traveling salesman problem. However, over the years, this approach (and its several variations, which are now collectively denominated *ant colony optimization* algorithms) has been applied to a wide variety of combinatorial optimization problems [108].

5 Conclusions

In this chapter, we have provided a short introduction to EAs and some of their applications within pattern recognition. This introduction included a summarized description of a few EAs (genetic algorithms, genetic programming and differential evolution). Then, several applications of them in segmentation, feature selection and classification (including clustering) were reviewed. This review indicated a great interest in using EAs for pattern recognition tasks, and also pointed out to the possible use of EAs combined with other approaches for the development of fully automated pattern recognition systems.

In the last part of the chapter, some further ideas to extend the research in this area were provided, including the use of other bio-inspired metaheuris-

tics and the incorporation of fitness approximation techniques to reduce the (normally high) computational cost associated to the use of EAs.

We really hope that the information provided in this chapter increases the interest from researchers working in pattern recognition to use EAs, for that has been its main goal.

Acknowledgements

The second author acknowledges support from CONACyT project no. 103570.

References

1. R.C. Gonzalez and R.E. Woods. *Digital Image Processing*. Addison Wesley, 1992.
2. K.R. Castleman. *Digital Image Processing*. Prentice Hall, 1996.
3. Sankar K. Pal and Paul P. Wang, editors. *Genetic Algorithms for Pattern Recognition*. CRC Press, Boca Raton, Florida, USA, 1996. ISBN 0-8493-9467-8.
4. M.M. Rizki, M.A. Zmuda, and L.A. Tamburino. Envolving pattern recognition systems. *IEEE Transactions on Evolutionary Computation*, 6(6):594–609, December 2002.
5. Fred Glover and Gary A. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer Academic Publishers, Norwell, Massachusetts, USA, 2003. ISBN 1-4020-7263-5.
6. Toshihide Ibaraki, Koji Nonobe, and Mutsunori Yagiura, editors. *Metaheuristics. Progress as Real Problem Solvers*. Springer, New York, USA, 2005. ISBN 0-387-25382-3.
7. David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Reading, Massachusetts, USA, 1989.
8. David B. Fogel. *Evolutionary Computation. Toward a New Philosophy of Machine Intelligence*. The Institute of Electrical and Electronic Engineers, New York, 1995.
9. A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, Berlin, 2003. ISBN 3-540-40184-9.
10. S.N. Sivanandam and S.N. Deepa. *Introduction to Genetic Algorithms*. Springer, Berlin, Germany, 2008. ISBN 978-3-540-73189-4.
11. Christian Blum and Andrea Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3):268–308, September 2003.
12. Colin B. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, Great Britain, 1993.
13. David 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.

14. John 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.
15. John H. Holland. Outline for a logical theory of adaptive systems. *Journal of the Association for Computing Machinery*, 9:297–314, 1962.
16. Hans-Paul Schwefel. Kybernetische evolution als strategie der experimentellen forschung in der strömungstechnik. Dipl.-Ing. thesis, 1965. (in German).
17. Hans-Paul Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*. Birkhäuser, Basel, Alemania, 1977.
18. Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. Wiley, Chichester, UK, 1981.
19. Lawrence J. Fogel. *Artificial Intelligence through Simulated Evolution*. John Wiley, New York, 1966.
20. Lawrence J. Fogel. *Artificial Intelligence through Simulated Evolution. Forty Years of Evolutionary Programming*. John Wiley & Sons, Inc., New York, 1999.
21. John R. Koza. Hierarchical genetic algorithms operating on populations of computer programs. In N. S. Sridharan, editor, *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 768–774. Morgan Kaufmann, San Mateo, California, 1989.
22. John R. Koza. *Genetic Programming. On the Programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge, Massachusetts, 1992.
23. John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. The MIT Press, Cambridge, Massachusetts, 1994.
24. John R. Koza, Forrest H. Bennet III, David Andre, and Martin A. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann Publishers, San Francisco, California, 1999.
25. Kenneth V. Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential Evolution. A Practical Approach to Global Optimization*. Springer, Berlin, 2005. ISBN 3-540-20950-6.
26. James Kennedy and Russell C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco, California, 2001.
27. Rui Xu and Don Wunsch. *Clustering*. IEEE Press and John Wiley & Sons, Hoboken, New Jersey, USA, 2009. ISBN 978-0-470-27680-8.
28. Guojun Gan, Chaoqun Ma, and Jianhong Wu. *Data Clustering: Theory, Algorithms, and Applications*. Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, USA, 2007. ISBN 978-0-898716-23-8.
29. J.B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley symposium on mathematical statistics and probability*, volume 2, pages 281–297, Berkeley, California, USA, 1967. University of California Press.
30. Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–249, May 2009.
31. M. Mahajan, P. Nimbhorkar, and K. Varadarajan. The planar k-means problem is NP-hard. *Lecture Notes in Computer Science*, 5431:274–285, 2009.
32. John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, 1975.
33. S. Ronald. *Genetic algorithms and permutation-encoded problems: Diversity preservation and a study of multimodality*. PhD thesis, The University of South Australia, 1995.

34. Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, New York, USA, third edition, 1996.
35. Franz Rothlauf. *Representations for Genetic and Evolutionary Algorithms*. Physica-Verlag, New York, 2002.
36. David E. Goldberg and Kalyanmoy Deb. A comparison of selection schemes used in genetic algorithms. In Gregory J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, San Mateo, California, 1991.
37. A. Kenneth De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Ann Arbor, Michigan, USA, 1975.
38. Lashon B. Booker. *Intelligent Behavior as an Adaptation to the Task Environment*. PhD thesis, Logic of Computers Group, University of Michigan, Ann Arbor, Michigan, USA, 1982.
39. A. Brindle. *Genetic Algorithms for Function Optimization*. PhD thesis, Department of Computer Science, University of Alberta, Edmonton, Alberta, Canada, 1981.
40. James Edward Baker. Reducing Bias and Inefficiency in the Selection Algorithm. In John 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.
41. J. J. Grefenstette and J. E. Baker. How Genetic Algorithms work: A critical look at implicit parallelism. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 20–27, San Mateo, California, June 1989. Morgan Kaufmann Publishers.
42. James Edward Baker. Adaptive Selection Methods for Genetic Algorithms. In John J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms*, pages 101–111. Lawrence Erlbaum Associates, Hillsdale, New Jersey, July 1985.
43. Gilbert Syswerda. Uniform Crossover in Genetic Algorithms. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9, San Mateo, California, 1989. Morgan Kaufmann Publishers.
44. Melanie Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, Cambridge, Massachusetts, 1996.
45. D. Dumitrescu, B. Lazzerini, L.C. Jain, and A. Dumitrescu. *Evolutionary Computation*. CRC Press, Boca Raton, Florida, USA, 2000. ISBN 0-8493-0588-8.
46. Bill P. Buckles and Frederick E. Petry, editors. *Genetic Algorithms*. Technology Series. IEEE Computer Society Press, 1992.
47. Günter Rudolph. Convergence Analysis of Canonical Genetic Algorithms. *IEEE Transactions on Neural Networks*, 5(1):96–101, January 1994.
48. A.E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, July 1999.
49. J.M. Peña, J.A. Lozano, and P. Larrañaga. An empirical comparison of four initialization methods for the k-means algorithm. *Pattern Recognition Letters*, 20:1027–1040, 1999.

50. U. Maulik and S. Bandyopadhyay. Genetic algorithm-based clustering technique. *Pattern Recognition*, 33:1455–1465, 2000.
51. K. Krishna and M Narasimha Murty. Genetic k-means algorithm. *IEEE Trans. on Sys. Man. & Cyber. B*, 29(3), 1999.
52. A. Fitzgibbon, M. Pilu, and R.B. Fisher. Direct least square fitting of ellipses. *IEEE Patt. An. & Mach. Intell.*, 21(5), May 1999.
53. S.J. Ahn, W. Rauth, and H-J. Warnecke. Least-squares orthogonal distances fitting of circle, sphere, ellipse, hyperbola, and parabola. *Pattern Recognition*, 34(12):2283–2303, Dec 2001.
54. L.G. de la Fraga, I. Vite Silva, and N. Cruz-Cortes. *Euclidean distance fit of conics using differential evolution*, volume 213, chapter Evolutionary Image Analysis and Signal Processing, pages 171–184. Springer, 2009.
55. L.G. de la Fraga and G.M. Lopez Dominguez. Robust fitting of ellipses with heuristics. In *2010 IEEE Congress on Evolutionary Computation, CEC 2010*, 2010. ACCEPTED.
56. F. Herrera, M. Lozano, and J.L. Verdegay. Tackling Real-Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis. *Artificial Intelligence Review*, 12(4):265–319, August 1998.
57. C. García-Martínez, M. Lozano, F. Herrera, D. Molina, and A.M. Sánchez. Global and Local Real-Coded Genetic Algorithms Based on Parent-Centric Crossover Operators. *European Journal of Operational Research*, 185(3):1088–1113, March 16 2008.
58. U.K. Chakraborty. *Advances in Differential Evolution. Studies in Computational Intelligence*. Springer, 2008.
59. Rainer Storn and Kenneth Price. Differential Evolution: A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. Technical Report TR-95-012, International Computer Science Institute, Berkeley, California, March 1995.
60. Rainer Storn and Kenneth Price. Differential Evolution - A Fast and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11(4):341–359, December 1997.
61. Hans-Paul Schwefel. *Evolution and Optimum Seeking*. John Wiley & Sons, New York, 1995.
62. K. Zielinski and R. Laur. Stopping criteria for differential evolution in constrained single-objective optimization. In U. K. Chakraborty, editor, *Advances in Differential Evolution. Studies in Computational Intelligence*. Springer, 2008.
63. Efrén Mezura-Montes, Jesús Velázquez-Reyes, and Carlos A. Coello Coello. Comparing Differential Evolution Models for Global Optimization. In Maarten Keijzer et al., editor, *2006 Genetic and Evolutionary Computation Conference (GECCO'2006)*, volume 1, pages 485–492, Seattle, Washington, USA, July 2006. ACM Press.
64. Kenneth V. Price. An Introduction to Differential Evolution. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 79–108. McGraw-Hill, London, UK, 1999.
65. Vitaliy Feoktistov and Stefan Janaqi. Generalization of the Strategies in Differential Evolution. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004), 2004, Santa Fe, New Mexico, USA*, page 165a, New Mexico, USA, April 2004. IEEE Computer Society.

66. S.M. Bhandarkar and H Zhang. Image segmentation using evolutionary computation. *IEEE Transactions on Evolutionary Computation*, 3(1):1–21, April 1999.
67. S. Kirkpatrick, C.D. Gellatt, and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.
68. M. Creutz. Microcanonical monte-carlo simulation. *Physical Review Letters*, 50(19):1411–1414, 1983.
69. Y.H. Wang, R.A. Prade, J. Griffith, W.E. Timberlake, and J. Arnold. A Fast Random Cost Algorithm for Physical Mapping. *Proceedings of the National Academy of Sciences of the United States of America*, 91(23):11094–11098, November 8 1994.
70. Pablo Moscato. Memetic Algorithms: A Short Introduction. In David Corne, Fred Glover, and Marco Dorigo, editors, *New Ideas in Optimization*, pages 219–234. McGraw-Hill, 1999.
71. Tianzi Jiang and Faguo Yang. An evolutionary tabu search for cell image segmentation. *IEEE Transactions on Systems, Man and Cybernetics Part B–Cybernetics*, 32(5):675–678, 2002.
72. Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston, Massachusetts, 1997.
73. J. Canny. A Computational Approach to Edge-Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, November 1986.
74. Leonardo Bocchi, Lucia Ballerini, and Signe Hässler. A New Evolutionary Algorithm for Image Segmentation. In Franz Rothlauf et al., editor, *Applications of Evolutionary Computing. Evoworkshops 2005: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, and EvoSTOC*, pages 264–273. Springer. Lecture Notes in Computer Science Vol. 3449, Lausanne, Switzerland, March/April 2005.
75. Martin Gardner. The fantastic combinations of John Conways new solitaire game "life". *Scientific American*, 223:120–123, October 1970.
76. James C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Kluwer Academic Publishers, Norwell, Massachusetts, USA, 1981. ISBN 0-306-40671-3.
77. Krzysztof Krawiec, Daniel Howard, and Mengjie Zhang. Overview of Object Detection and Image Analysis by Means of Genetic Programming Techniques. In *Proceedings of the 2007 Frontiers in the Convergence of Bioscience and Information Technologies*, pages 779–784. IEEE Computer Society Press, October 11–13 2007.
78. Durga Prasad Muni, Nikhil R. Pal, and Jyotirmoy Das. Genetic Programming for Simultaneous Feature Selection and Classifier Design. *IEEE Transactions on Systems, Man, and Cybernetics Part B–Cybernetics*, 36(1):106–117, February 2006.
79. Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, New York, second edition, September 2007. ISBN 978-0-387-33254-3.
80. Ukrit Watchareeruetai, Yoshinori Takeuchi, Tetsuya Matsumoto, Hiroaki Kudo, and Noboru Ohnishi. Transformation of redundant representations of linear genetic programming into canonical forms for efficient extraction of image features. In *2008 IEEE Congress on Evolutionary Computation (CEC 2008)*, pages 1996–2003, Hong Kong, June 2008. IEEE Service Center.

81. Markus F. Brameier and Wolfgang Banzhaf. *Linear Genetic Programming.* Springer, New York, USA, 2007. ISBN 978-0-387-31029-9.
82. Taras Kowaliw, Wolfgang Banzhaf, Nawwaf Kharma, and Simon Harding. Evolving Novel Image Features Using Genetic Programming-based Image Transforms. In *2009 IEEE Congress on Evolutionary Computation (CEC'2009)*, pages 2502–2507, Trondheim, Norway, May 2009. IEEE Press.
83. J. F. Miller, P. Thomson, and T. Fogarty. Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study. In D. Quagliarella, J. Périaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science*, pages 105–131. Morgan Kaufmann, Chichester, England, 1998.
84. Julian F. Miller and Peter Thomson. Cartesian Genetic Programming. In Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian Miller, Peter Nordin, and Terence C. Fogarty, editors, *Genetic Programming, European Conference, EuroGP 2000*, pages 121–132, Edinburgh, Scotland, UK, April 2000. Springer. Lecture Notes in Computer Science Vol. 1802.
85. Pei-Fang Guo, Prabir Bhattacharya, and Nawwaf Kharma. An Efficient Image Pattern Recognition System Using an Evolutionary Search Strategy. In *Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics*, San Antonio, Texas, USA, October 2009. IEEE Press.
86. Tom M. Mitchell. *Machine Learning.* McGraw-Hill, USA, 1997.
87. Vladimir N. Vapnik. *The Nature of Statistical Learning Theory.* Springer, second edition, 1999. ISBN 978-0387-98780-4.
88. Michael L. Raymer, William F. Punch, Erik D. Goodman, Leslie A. Kuhn, and Anil K. Jain. Dimensionality Reduction Using Genetic Algorithms. *IEEE Transactions on Evolutionary Computation*, 4(2):164–171, July 2000.
89. Beatriz de la Iglesia, Alan Reynolds, and Vic J Rayward-Smith. Developments on a Multi-objective Metaheuristic (MOMH) Algorithm for Finding Interesting Sets of Classification Rules. In Carlos A. Coello Coello, Arturo Hernández Aguirre, and Eckart Zitzler, editors, *Evolutionary Multi-Criterion Optimization. Third International Conference, EMO 2005*, pages 826–840, Guanajuato, México, March 2005. Springer. Lecture Notes in Computer Science Vol. 3410.
90. B. de la Iglesia, G. Richards, M.S. Philpott, and V.J. Rayward-Smith. The application and effectiveness of a multi-objective metaheuristic algorithm for partial classification. *European Journal of Operational Research*, 169:898–917, 2006.
91. Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.
92. D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz. UCI Repository of machine learning databases, 1998. http://www.ics.uci.edu/∼mlearn/MLRepository.html.
93. Rahila H. Sheikh, M.M. Raghuwanshi, and Anil N. Jaiswal. Genetic Algorithm Based Clustering: A Survey. In *First International Conference on Emerging Trends in Engineering and Technology*, pages 314–319, Nagpur, Maharashtra, India, July 2008. IEEE Press.
94. Sanghamitra Bandyopadhyay and Ujjwal Maulik. Genetic clustering for automatic evolution of clusters and application to image classification. *Pattern Recognition*, 35(6):1197–1208, June 2002.

95. D.L. Davies and D.W. Bouldin. Cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(2):224–227, 1979.
96. Sanghamitra Bandyopadhyay, Sankar K. Pal, and B. Aruna. Multiobjective GAs, Quantitative Indices, and Pattern Classification. *IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics*, 34(5), October 2004.
97. Joshua D. Knowles and David W. Corne. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation*, 8(2):149–172, 2000.
98. Ranajit Das, Sushmita Mitra, Haider Banka, and Subhasis Mukhopadhyay. Evolutionary Biclustering with Correlation for Gene Interaction Networks. In Ashish Ghosh, Rajat K. De, and Sankar K. Pal, editors, *Pattern Recognition and Machine Intelligence. Second International Conference (PReMI'2007)*, pages 416–424. Springer, Lecture Notes in Computer Science, Vol. 4815, Kolkata, India, December 18-22 2007. ISBN 978-3-540-77045-9.
99. Paulo V. W. Radtke, Tony Wong, and Robert Sabourin. Solution Over-Fit Control in Evolutionary Multiobjective Optimization of Pattern Classification Systems. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(6):1107–1127, September 2009.
100. Clement Chatelain, Sebastien Adam, Yves Lecourtier, Laurent Heutte, and Thierry Paquet. A multi-model selection framework for unknown and/or evolutive misclassification cost problems. *Pattern Recognition*, 43(3):815–823, March 2010.
101. Yaochu Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, 9(1):3–12, 2005.
102. David Corne, Marco Dorigo, and Fred Glover, editors. *New Ideas in Optimization*. McGraw-Hill, London, 1999.
103. James Kennedy and Russell 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.
104. Dipankar Dasgupta, editor. *Artificial Immune Systems and Their Applications*. Springer-Verlag, Berlin, 1999.
105. Leandro Nunes de Castro and Jonathan Timmis. *Artificial Immnue System: A New Computational Intelligence Approach*. Springer Verlang, Great Britain, September 2002. ISBN 1-8523-594-7.
106. Wei Wang, Shangce Gao, and Zheng Tang. Improved pattern recognition with complex artificial immune system. *Soft Computing*, 13(12):1209–1217, October 2009.
107. M. Dorigo and G. Di Caro. The Ant Colony Optimization Meta-Heuristic. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, UK, 1999. McGraw-Hill.
108. Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. The MIT Press, 2004. ISBN 0-262-04219-3.

Index