Evolutionary Prototyping for Handwriting Recognition

L. P. Cordella
Dipartimento di Informatica e Sistemistica
Università di Napoli Federico II,
Via Claudio, 21 80125 Napoli – ITALY
cordel@unina.it

C. De Stefano
Dipartimento di Automazione, Elettromagnetismo, Ingegneria dell’Informazione e Matematica Industriale
Università di Cassino
Via G. Di Biasio, 43 02043 Cassino (FR) – ITALY
destefan@unicas.it

F. Fontanella
Dipartimento di Informatica e Sistemistica
Università di Napoli Federico II,
Via Claudio, 21 80125 Napoli – ITALY
frfontan@unina.it

A new prototyping method based on the evolutionary computation paradigm and on the concept of Vector Quantization is proposed. It uses a specifically devised evolutionary algorithm for evolving a set of prototype feature vectors and does not require any a-priori knowledge about neither the actual number of prototypes nor the statistical properties of the input data. Experiments performed by using both synthetic data and handwritten digits randomly extracted from the NIST database have confirmed the effectiveness of the approach.

Keywords: Evolutionary Computation, Pattern Recognition

1. Introduction

Prototyping techniques play an important role in many pattern recognition and computer vision tasks and have been widely used in the context of handwriting recognition. In this framework, prototype definition represents a crucial step of the design, in that the set of prototypes should satisfy two conflicting requirements: completeness, in order to guarantee that prototypes based on the training samples apply to the broadest possible future test set, and consistency, so as to ensure that the system will be able to correctly predict the class to which the unknown samples belong to. In other words, completeness guarantees that the system is able to dealing with the variability among specimens belonging to the same class produced by
different writers or by the same writer at different times, while consistency ensures that the system will discriminate among similar samples belonging to different classes. Moreover, due to the extreme variability exhibited by samples produced by a large population of writers, achieving such an aim often requires the use of a large number of prototypes for each class, in order to capture the distinctive features of different writing styles, and rather complex classification algorithms. Obviously, the higher the number of prototypes, the higher the computational cost during classification.

Independently of the particular considered technique, the most relevant problem that any prototyping method needs to deal with, is the determination of the actual number of prototypes. In fact, the variability of writing styles generated by a large number of writers, generally implies that it is very difficult to a priori determine the number of different ways in which a character may be written. As a consequence it is very difficult to predict the number of different prototypes that should be used to properly describe each class. When the number of prototypes is underestimated, obviously some of them will not be generated and the corresponding samples not recognized. When the number of prototypes is overestimated, some unnecessary prototypes may be generated exhibiting a certain degree of similarity with samples of different classes: this may lead to ascribe similar samples, but belonging to different classes, to the wrong class thus reducing the performance of the system.

An effective way to face this problem may be that of using Evolutionary Algorithms (EAs) for automatically inferring the prototypes from a training set of examples. Such algorithms, in fact, (see Section 2 for a general overview) combine a powerful tool for finding solutions in complex high–dimensional search spaces, with an explicit mechanism for verifying the effectiveness of the prototypes. Thanks to their ability to solve hard problems characterized by complex and high dimensional search spaces, Evolutionary Algorithms have been successfully used to solve classification problems. A short review of previous work in this field will be presented in Section 3, after a propaedeutical overview of the evolutionary approach (Section 2).

In this framework, we propose a new prototyping method, also founded on the concept of Vector Quantization. It uses a specifically devised evolutionary algorithm for evolving a set of feature vectors and does not require any a-priori knowledge about neither the actual number of prototypes nor the statistical properties of the input data. The effect of Vector Quantization, in both classical pattern recognition and neural network fields, is a Voronoi tessellation of the feature space in which data points are located. In practice, Vector Quantization allows to partition the feature space into a number of regions, each identified by a reference vector. Such regions are bordered by hyperplanes (line segments in a two dimensional space) defined as the loci of points that are equidistant from the two nearest reference vectors. The reference vectors represent the centroids of the corresponding regions and each region represents a cluster.

The proposed prototyping method has been used for implementing a hand-
written digit recognition system. In this application domain, the actual number of clusters in the feature space is generally a-priori unknown because the shape variability of the samples produced by different writers implies that each class is actually made of a number of subclasses. The data to be classified are the samples of the training set and the results of the prototyping process is a set of reference vectors (prototypes), each of which is assigned the label of the training set samples which are more frequent in the corresponding region. Once the set of prototypes has been obtained, classification is performed by assigning to an unknown sample the label of the nearest prototype in the feature space. Experiments performed by using both synthetic data and handwritten digits randomly extracted from the NIST database have shown very promising results and have confirmed the effectiveness of the approach.

The remainder of the paper is organized as follows: Section 2 describes the evolutionary computation paradigm. Section 3 reviews some previous evolutionary approaches to classification. Section 4 illustrates the proposed prototyping technique, while in Section 5 the experimental results are presented. Eventually, Section 6 is devoted to the conclusions.

2. Evolutionary Algorithms

In the natural world, evolution is mostly determined by natural selection caused by the competition among different individuals for the resources available. In this competition, the fittest individuals are more likely to survive and propagate their genetic material through future generations. This natural phenomenon has been widely studied by computer scientists since the last years of the 50's of the last century (3; 5; 4). Those scientists guessed that, as natural evolution has been able to evolve highly complex structures such as plants and animals, algorithms simulating this natural process could be devised in order to solve problems requiring complex and hard to find solutions. The result of this insight has been a new computation paradigm, called Evolutionary Computation (EC in the following), largely used to implement several different algorithms, usually called Evolutionary Algorithms (EAs in the following).

The EC paradigm is based on a small set of essential elements that rule evolution by natural selection: reproduction of individuals, variation phenomena that affect the likelihood of survival of individuals, inheritance of many of the parents’ features by offspring and the presence of a finite amount of resources causing competition for survival among individuals. These simple features – reproduction, likelihood of survival, variation, inheritance and competition – are the bricks that build the simple model of evolution employed by computer scientists to define the EC paradigm. This model has demonstrated to be able to solve difficult problems (6), so as NP-hard optimization problems, in which optimal solutions have to be found amongst a very huge number of possible solutions, or machine learning problems, in which solutions are represented by complex models able to represent objects belonging to
different classes.

More specifically, EC–based algorithms find solutions for a given problem by generating a population of individuals, i.e. a set of tentative solutions. Then the “goodness” of each individual as solution of the problem is evaluated by means of a fitness function containing all the knowledge on the problem to be solved. Finally, a new population is generated by selecting individuals in the current population. The selection process is stochastic, but favors individuals having higher fitness values with respect to those which have lower fitness values. Afterwards, selected individuals can be modified, according to a given probability, by variation operators, in order to generate, if possible, new and better, individuals. This process is repeated until one or more conditions are not satisfied. Formally, given a population \( p_t \) at time \( t \), an evolutionary algorithm applies the variation operators \( v \) to the population. Variations are applied according to a selection method \( s \), where individuals compete to be selected according to their fitness. A population \( p_{t+1} \) at time \( t + 1 \) is found by:

\[
p_{t+1} = v(s(p_t))
\]

The variation operators provide new solutions by modifying the existing ones. In evolutionary algorithms, these operators usually are recombination and mutation. In the EC jargon the individuals chosen by the selection method and given as input to the variation operators are called parents, whereas the individuals generated by these operators as output are called offspring. The use of a population of solutions, together with the variation operators and a selection mechanism, implementing competition among individuals, provides an effective strategy to explore the solution space of the problem to be solved. Note that only those solutions which are reachable by the operators, i.e. those obtainable by iteratively applying the operators to the individuals of the initial population \( (p_0) \), could be visited during the evolutionary process or run. The basic outline of an Evolutionary Algorithm is schematically illustrated in Fig. 1.

Due to the generality of the computational scheme just described, many EC–based algorithms have been proposed since its first appearance, and probably many others will be proposed in the future. Given a problem to be solved, an evolutionary algorithm can be designed for that problem after defining the following four elements:

- A solution encoding, i.e. a data structure suitable for encoding the solutions of the problem at hand. The characteristics of such structure should make the implementation of variation operators easy.
- A set of variation operators that produce offspring by modifying the selected parent individuals. The operators implement the concept of inheritance through stochastic variations and are strictly related to the data structure employed to represent the solutions of the problem at hand.
- A fitness function that evaluates each individual and assigns to it a score,
or fitness value.

- A Selection method that implements a choice mechanism that favors individuals with higher fitness.

These key elements of any EC–based algorithm will now be discussed in detail.

2.1. Solution Encoding

The operators employed and the different representations adopted for encoding the solutions are commonly used in order to categorize the EAs into four main groups: genetic algorithms (GAs) use a bit-string and two-parent crossover (8), evolutionary strategies use a real-valued vector and Gaussian mutation (14), evolutionary programming employs a finite-state machine and mutation operators (15), and genetic programming (GP) uses a computer program or executable structure and two-parent crossover (16).

Besides the above four ways of encoding solutions, many specific encoding techniques have been developed in order to solve particular problems. Among the others, grouping problems and graph generation problems have induced the development of specific encodings (19; 20). New methods, involving specific representation, have also been proposed in the fields of molecular design (21). Moreover, different specific encoding schemes have been also used for evolving artificial neural networks (22).

begin
  randomly initialize a population of $P$ individuals;
  evaluate the fitness of each individual;
  while (termination criterion is not fulfilled) do
    copy the best $E$ individuals in the new population;
    for $i = 0$ to $(P - E)/2$ do
      apply the selection mechanism;
      replicate the selected individuals;
      if flip($p_c$) then
        apply the crossover operator on the selected individuals;
      end if
      perform mutation on the offspring;
      evaluate the fitness of each individual;
    end for
    replace the old population with the new one;
    update variables for termination;
  end while
end

Fig. 1. The outline of an Evolutionary Algorithm. The function flip($p$) returns the value 1 with probability $p$ and the value 0 with a probability $(1 - p)$. 

2.2. Operators

The role of the variation operators, also called genetic operators, from a search standpoint, is that of exploring new areas of the solution space, searching for new and possibly better solutions. The search is usually performed by randomly modifying the individual(s) given in input. Operators are usually characterized by a quantity $p$ which indicate the probability according to which an operator is applied to one or more individuals. Hence, there is a probability $(1 - p)$ that such individuals remain unchanged. Note that the higher $p$, the higher the exploration of the search space.

Genetic operators can be grouped in two main classes, recombination and mutation. The first class groups those operators that take as input two or more individuals and give as output the same, or even lower, number of individuals. The effect of the recombination operators is that of swapping or combining parts of the individuals given in input. The effects of this kind of operators strongly depend on the similarity degree of the individuals it is applied to [Sha01]. In fact, the more similar the input individuals, the lesser the degree of diversity between input and output individuals will be. As regards mutation operators, instead, they take as input a single individual and yield as output a new individual obtained by randomly modifying the input one. The effect of this operator is that of a random exploration of the neighborhood of the input individual.

2.3. Selection Methods

Selection methods are based on stochastic mechanisms that make it possible to choose, by using fitness–related information in a probabilistic manner, the individuals to be used as parents for producing offspring in the next population. The role of selection is that of exploiting the information acquired so far in order to find better solutions. Such exploitation takes place favoring individuals having better fitness. Nevertheless, individuals having lower fitness are not completely excluded from the choice process. Selection methods are characterized by some quantities that essentially measure the degree of exploitation of the information held by individuals that have a good fitness. If $P(t + 1)$ is the population generated from the population $P(t)$ by a given selection method $\sigma$, the following two quantities can be defined:

- **Selection intensity**: If $\bar{f}_{P(t+1)}$ is the expected average fitness value of $P(t + 1)$ and $\bar{f}_{P(t)}$ is the expected average fitness value of $P(t)$, the selection intensity is given by

$$S_{\sigma} = \frac{\bar{f}_{P(t+1)}}{\bar{f}_{P(t)}}$$

- **Loss of diversity**: Percentage of individuals in the population $P(t)$ that have not been selected by $\sigma$ and then are not present in $P(t + 1)$. 


From a search perspective, the selection intensity measures the degree of exploitation of the available information (the fitness values of the individuals in the current population). The loss of diversity, instead, measures the loss of information due to the selection process. These quantities are directly proportional: the higher the selection intensity, the higher the loss of diversity. Note that if too high values are used for such quantities, the system tends to perform like a greedy heuristic. On the contrary, if too small values are used the system behavior is similar to that of random search algorithms.

2.4. Fitness

As mentioned above, the fitness function measures the “goodness” of an individual as solution of the problem at hand and may be a simple mathematical function or its computation may be as complex as running an elaborate simulation. In practice, such a function contains the whole available knowledge on the problem. This information is exploited by the selection mechanism in order to choose the individuals that will undergo genetic manipulation for producing the new population. Looking at this fact from a search space standpoint, the fitness function provides information for locating regions containing good solutions. The fitness function can be seen as the “driver” of the search performed by any EC–based technique, that guides the search toward those regions of the solution space that seem to be more promising, according to the built-in knowledge of the fitness function. As a consequence the definition of an appropriate fitness function for the problem faced, is a key issue in the designing phase of any EC–based algorithm.

3. Evolutionary Approaches to Classification

As already said, the use of Evolutionary Algorithms for solving classification problems has often been attempted. Specifically, GAs have been applied to evolve sets of rules for more than two decades. These rules predict the class of a sample specifying some values of the sample attributes. This methodology forms a machine learning paradigm called learning classifier systems (LCS) (23). In this approach each individual encodes one or more rules of the form IF-THEN: the rule antecedent (the IF part) contains a combination of conditions on some attribute values, while the rule consequent (the THEN part) expresses the class predicted by the rule. GAs for rule discovery can be divided into two main classes, called Michigan and Pittsburgh, based on how rules are encoded by individuals. In the Michigan (24; 25) approach each individual encodes a single prediction rule, whereas in the Pittsburgh approach (26; 27; 28) each individual encodes a whole set of prediction rules.

The Pittsburgh approach was originally devised to solve single–class problems and then only the antecedent part of a rule is encoded, samples that match one or more rules are classified as a positive examples of the class, whereas samples that do not match any rule are classified as a negative examples. In order to tackle multi–class problems, they have been extended by introducing multiple populations.
so that each population is dedicated to learn rules for a specific class. Note that it is also possible that a sample is matched by more rules belonging to different classes or by any rules of any class. Unfortunately, this problem has not been addressed in many of the systems based on the Pittsburgh approach.

Also the Michigan approach was developed to cope with problems for a single-class only, as well. In case of multi-class problems, these algorithms are run once for each class, where each run evolves a set of rules for a specified class. So as in the Pittsburgh approach, it is possible that an instance is matched by several rules, each predicting a different class, or it is also possible that an instance is matched by none of any rule predicting any class. In order to overcome this problem affecting both the approaches, some hybrid Michigan/Pittsburgh methods have also been proposed (29).

More recently some attempts have been done to cope with classification problems by using Genetic Programming (30; 31; 32; 33). In (31), GP has been used to evolve equations involving simple arithmetic operators and feature variables, for hyper-spectral image classification. In (30), GP has also been employed for image classification problems, adding exponential functions, conditional functions and constants to the simple arithmetic operators. In (34) GP has been used to generate discriminant functions which carry out arithmetic functions with fuzzy attributes. In (39) GP has been used to evolve populations of fuzzy rule sets, whereas a simple evolutionary algorithm was employed to evolve the membership function definitions. The populations involved are allowed to co-evolve in such a way that both rule sets and membership functions can adapt each other. In (33), an interesting method which considers a $c$-class problem as a set of $c$ two-class problems has been introduced. When the expression for a particular class is searched, that class is considered as target, while the other ones are merged and treated as a single undesired class. In this way, $c$ expressions can be obtained performing $c$ runs. These expressions can then be used concurrently for discriminating the $c$ different classes of the problem at hand. In all the above quoted approaches, the number $c$ of classes to be dealt with is used to divide the data set at hand in exactly $c$ parts. Thus, these approaches require that the number of actual classes is a priori known or, in other words, do not take into account the existence of subclasses within one or more of the classes in the analyzed data set.

In the EC field, classification tasks have also been faced by considering them as multimodal optimization problems: a single prototype is seen as one of the several solutions to be searched for (35; 36). In this kind of approach the individuals representing the candidate prototypes are concurrently evolved in the same population and niching methods (37; 38) are used to form and maintain the subpopulations the searched solutions may originate from. These methods consider a good solution (i.e. a prototype) and its neighborhood in the solution space as a niche. The aim of these algorithms is that of creating groups of individuals, usually called subpopulations, each one occupying a different niche. Finding a subpopulation allows us to searching for the best solution within the niche it occupies. Among niching meth-
ods, fitness sharing methods are the best known and the most widely used (36). In this approach, a distance measure is defined and the individuals having distances among them lower than a given value (niche radius) are considered as belonging to the same niche. Such individuals are forced to share a given amount of fitness. In practice, the sharing is obtained by reducing the fitness of all the individuals within the niche according to their number. The effective use of the fitness sharing in real world problems, however, is severely limited because it requires a high degree of knowledge about the fitness function landscape, while for most of such problems such knowledge is hard to achieve.

4. Our Approach

The method proposed in this paper for finding a good set of prototypes is based on a particular class of genetic algorithms, namely the Breeder Genetic Algorithms (BGAs) (17), in which the individuals are encoded as real valued vectors. In our case, an individual consists of a variable length list of prototypes, also referred in the following as genes, each one represented by a centroid in the feature space, (see Figure 2). The number of genes will be referred as length. We also assume that an individual is made up by the whole set of prototypes. The knowledge about the specific problem to be solved is limited to the choice of a training set $D_{tr}$ of labeled samples.

The above assumption offers several advantages. First of all it makes it possible to transform a multi-objective problem into a single-objective one, overcoming the drawback of discovering and maintaining a population containing all the solutions, as discussed in the previous Section. The second advantage is that a simple and effective fitness function can be defined. In fact, in our approach, the fitness of an individual depends on the recognition rate obtained by using for classification the whole set of prototypes making up that individual. Moreover, in order to find the minimum number of prototypes, a term that favors individuals having a lower number of prototypes, is added to the fitness function. The third advantage is that it is possible to evolve individuals made of a variable number of prototypes. As discussed in the Introduction, discovering the actual number of prototypes needed to represent the samples of a given class is a crucial issue in the design of an handwriting recognition system.

4.1. Evolutionary Learning of Prototypes

In the initial population, each of the $P$ individuals is generated in the following way:

i) first, a number $n$ is randomly chosen in the range $[N_{\text{min}}, N_{\text{max}}]$, with a uniform probability distribution. While $N_{\text{min}}$ is usually chosen equal to the number of a priori classes of the problem at hand, the value of $N_{\text{max}}$ has to be chosen by the user and represents a parameter of the algorithm. The number $n$ represents the number of prototypes making up the $i$–th individual.
ii) Then, \( n \) prototypes are randomly generated.

After the initial population has been created, the prototypes of each individual are labeled by using a dynamic labeling mechanism that will be discussed in the next Section. Then, the fitness of each individual is evaluated. Afterwards a new population is generated. The first \( \mathcal{E} \) individuals with higher fitness values are just copied in the new population. This realizes an elitist strategy\(^a\). In order to complete the next generation, any selection mechanism can be used for choosing the remaining \( (P - \mathcal{E})/2 \) couples of individuals. To every selected couple, a recombination operator is applied according to a chosen probability factor \( p_c \). Then, a mutation operator is applied according to a probability factor \( p_m \). The so built new population replaces the old one and the process is repeated for \( N_g \) generations. The fitness function, the selection mechanism and the operators employed will be described in the following.

**Fitness function** Each individual is evaluated using a training set \( \mathcal{D}_{tr} \) containing \( N_{tr} \) samples. This evaluation implies the following steps:

1. The prototypes are labeled by using the procedure described in Section 4.2.
2. The recognition rate obtained on \( \mathcal{D}_{tr} \) is computed and assigned as fitness value to the individual.

In order to favor the individuals able to obtain good performances with a smaller number of genes, the fitness of each individual is increased by \( 0.1/l \), where \( l \) is the length of the individual. Moreover, the individuals having a length out of the interval \([N_{\text{min}}, N_{\text{max}}]\) are killed, i.e. marked in such a way that they are never more chosen by the selection mechanism.

**Selection mechanism** The tournament method has been chosen as selection mechanism. In the tournament selection, \( T \) individuals are randomly chosen from the population and the best individual of this group is selected as

\(^a\)An elitist strategy ensures that the fittest members of the population are passed on to the new one, without being altered by genetic operators. Using elitism ensures that the best individuals so far evolved can never been lost from one generation to the next. From a search space perspective, such a mechanism preserves the information contained in the best area so far located.
Parents

Offspring

Fig. 3. An example of application of the crossover operator.

parent. Such a mechanism ensures to control the loss of diversity and the selection intensity (18)

**Genetic operators** The used operators are: crossover and mutation. The crossover operator is applied to two individuals $I_1$ and $I_2$ and yields two new individuals by swapping parts of the lists of the initial individuals, without breaking any single gene (see Figure 3). Assuming that the length of $I_1$ and $I_2$ are respectively $l_1$ and $l_2$, the crossover is applied in the following way: the first individual is split in two parts by randomly choosing an integer $t_1$ in the interval $[1, l_1]$. The obtained lists of vectors $I_1'$ and $I_1''$ will have length $t_1$ and $l_1 - t_1$ respectively. Analogously, by randomly choosing an integer $t_2$ in the interval $[1, l_2]$, two lists of vectors $I_2'$ and $I_2''$, respectively of length $t_2$ and $l_2 - t_2$, are obtained from $I_2$. At this stage, in order to obtain a new individual, the lists $I_1'$ and $I_2'$ are merged. This operation yields a new individual of length $t_1 + l_2 - t_2$. The same operation is applied to the remaining lists $I_2'$ and $I_1''$ and a new individual of length $l_2 + l_1 - t_1$ is obtained. The number of swapped genes depends on the integers $t_1$ and $t_2$. It is worth noting that the implemented crossover operator allows us to obtain individuals of variable length. Hence, during the evolution process, individuals made of a variable number of prototypes can be evolved.

Given an individual $I$, the mutation operator is independently applied to each gene $G_i$ of $I$ in the following way:

```plaintext
begin
  for $j = 0$ to $N_F$ do
    range = rndreal(0.1 * $\delta_j$)
    if flip($p_m$) then
      $G_i[j] = G_i[j] \pm$ range (+ or - with equal probability);
    end if
  end for
end
```

Where $N_F$ is the number of features in a vector, $\delta_j$ is the range of the $j$–th
feature computed on the training set, and $p_m$ represents the probability of mutation of each single feature value in the gene.

4.2. Dynamic Labeling of the Prototypes

In the approach we propose, prototypes making up an individual are not a priori labeled, but their labeling occurs after each sample in the training set has been assigned to its nearest prototype. In the following, the procedure employed for labeling the prototypes, called dynamic labeling, is given:

1. Every sample in the training set is assigned to the nearest gene (i.e. prototype) in the individual to be evaluated. Euclidean distance is used in the feature space. After this step, $p_i \ (p_i \geq 0)$ samples will have been assigned to the $i$-th prototype. In the following the prototypes for which $p_i > 0$ will be referred to as valid. The remaining prototypes ($p_i = 0$) will be ignored.

2. Each valid prototype is labeled with the label most widely represented among the patterns that have been assigned to it.

In figure 4 an example of application of this procedure is given. In the example, four classes have been defined and the individual is made up of four prototypes. The first prototype $p_1$ is labeled with the label of the fourth class, as it has demonstrated to be a good prototype for that class, while, for the same reason, $p_2$ has been labeled with the label of the first class. Prototypes $p_3$ and $p_4$ have respectively been labeled with the labels of the third and second class.

Note that the dynamic labeling, together with the recombination operator previously described, allow the method to automatically finding the most proper number of prototypes for the considered classification problem. Yet, prototype dynamic labeling allows us to relax a strong constraint due to the a priori labeling of the prototypes (see figure 4). In fact, suppose that the analyzed data contains $c$ classes

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>3</td>
<td>38</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$c_2$</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>35</td>
</tr>
<tr>
<td>$c_3$</td>
<td>0</td>
<td>0</td>
<td>47</td>
<td>3</td>
</tr>
<tr>
<td>$c_4$</td>
<td>60</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
</tbody>
</table>

Fig. 4. A comparison between the prototype dynamic labeling of an individual made of 4 prototypes (a) and the a priori labeling of the same labeling (b). In (b) the prototypes are labeled according to their position within the list making up the individual.
and each individual also contains c a priori labeled prototypes, the constraint imposed on the labels of the prototype reduces of a factor (c!) the number of solutions to be considered as good solutions of the problem at hand. In fact, among the (c!) possible permutations of a set of c prototypes, each representing a different class, just one is considered a good solution, while the remaining ones are considered bad solutions. This occurs because, in the discarded solutions, the position of a given prototype in the list may not coincide with the label preliminarily assigned to it (see Figure 4(b)). Considering the example given in Figure 4, if the dynamic labeling is used and the individual is evaluated according to the fitness function previously described (without taking into account the term evaluating the length of an individual), then the value 0.9 is assigned to the individual as fitness value. While, if the prototypes of the individual are a priori labeled, then the value 0.32 is assigned as fitness value. In fact, the first prototype should be labeled with the first label despite it is a good prototype for the fourth class, while, on the contrary, the second prototype should be labeled with the second label despite it is a good prototype for the first class. Such constraint becomes stronger as c increases. For example, if we assume that 20 classes are a priori defined for the data, and a set of 20 good prototypes is given, then just one solution, (i.e. one individual) containing this set has a good fitness, while the other ≈ 2 · 10^18 solutions, made of permutations of the same prototypes, have a significantly smaller fitness value.

5. Experimental Results

Two sets of experiments have been carried out to assess the effectiveness of the proposed approach. The first set has regarded four data sets that have been synthetically generated using mathematical distributions. The aim of this set of experiments has been that of obtaining a first evaluation of the effectiveness of the proposed approach. The second set of experiment has been carried out in order to evaluate the effectiveness of the approach in handwriting recognition. This set of experiments has regarded a data set, belonging to the National Institute of Standards and Technology (NIST) (41) in which each sample represents an handwritten digit. In the following the method used for the normalization of the analyzed data, the settings of the basic evolutionary parameters, and the results obtained on both sets of experiments are reported.

5.1. Normalization and Parameter Settings

The features of the samples of the data sets taken into account have been normalized in the range [−1.0, 1.0]. Given a not normalized sample $x = (x_1, \ldots, x_N)$, every

---

"An obvious way of a priori labeling the prototypes is that of labeling the first one with the label of the first class, the second one with the label of the second class and so on."
feature \( x_i \) is normalized using the formula:

\[
x_i = \frac{x_i - \overline{x}_i}{3\sigma_i}
\]

where \( \overline{x}_i \) and \( \sigma_i \), respectively represent the mean and the standard deviation of the \( i \)-th feature computed over the whole data set.

The basic parameters of the implemented evolutionary algorithm (described in Section 4) that have been used in all the experiments are summarized in the following Table:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>symbol</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>( \mathcal{P} )</td>
<td>300</td>
</tr>
<tr>
<td>Tournament size</td>
<td>( \mathcal{T} )</td>
<td>7</td>
</tr>
<tr>
<td>Elitism size</td>
<td>( \mathcal{E} )</td>
<td>5</td>
</tr>
<tr>
<td>Crossover probability</td>
<td>( p_c )</td>
<td>0.4</td>
</tr>
<tr>
<td>Mutation probability</td>
<td>( p_m )</td>
<td>0.08</td>
</tr>
<tr>
<td>Number of Generations</td>
<td>( N_g )</td>
<td>500</td>
</tr>
</tbody>
</table>

Since EA are stochastic algorithms, 20 runs have been performed for each data set taken into account. In case of synthetic data, the reported results are those obtained using the individual having the highest fitness value among those obtained during the 20 performed runs. In case of handwritten characters, for comparison purposes, the reported results are those obtained by averaging the recognition rates achieved on the test set by the best individual evolved at each of the 20 runs.

5.2. Synthetic Data

The first test has been performed on four data sets synthetically generated. Each data set contains 5000 samples, equally distributed in five classes. Each sample is a feature vector consisting of 2 real elements. As regards the minimum number (\( N_{\text{min}} \)) and the maximum number (\( N_{\text{max}} \)) of genes allowed in an individual, they have been set respectively to 5 (the number of a priori classes or, in other words, of labels to be assigned) and 20. From each of the four data sets considered, two statistically independent sets of data, a training set and a test set, have been randomly extracted. The former has been used to evaluate the fitness of the individuals, as reported in Section 4, the latter to evaluate the performance of our system on unknown data. In Figure 5 the four synthetically generated data sets and the prototypes, i.e. the genes of the best individual found out by the system, are displayed. In this figure, the best recognition rates on test sets are also reported. Let us now discuss the results obtained on the four data sets. The data sets displayed in Figure 5 contain samples whose features are respectively in the ranges \([-25.0, 25.0]\) and \([-30.0, 20.0]\). Each of these data sets contains five clusters. Some of them are more or less elongated, while the remaining one has circular shape. Some clusters are well separated, while others are very close or crossing. In three case the system has been able to find the
minimum number to find the minimum number of prototypes needed, i.e. 5. Moreover, for the close clusters the prototypes have been located in such a way to have a quite high recognition rate. In one case, (Figure 5(c)) the system has found 7 prototypes. This higher number of prototypes is due to the fact that for each of the crossing clusters two prototypes are needed. In this case the best recognition rate achieved on the test set has been a little lower.

5.3. Real Data

The second set of experiments has been performed on digits extracted from NIST database. Each item of this data set is a 64×64 binary image (a bitmap of 4096 pixels), representing an handwritten digit. From this database two statistically independent sets of data, a training set and a test set, have been randomly extracted. Each set is made of 10,000 samples (1,000 per class).

Let us briefly summarize the adopted character description method, i.e. the procedure through which the input bitmap is converted into a form compatible with the Breeder Genetic Algorithm used in our approach. The description of each
sample of the data set has been obtained by carrying out the following steps:

1. The original 64×64 pixels bitmap image is divided into 64 parts, each consisting of 8×8 pixels.
2. For each of the part obtained in the previous step, the value of the average gray level is computed on the 64 pixels.

According to the description method adopted each sample in the data set is represented by a feature vector of 64 real elements in the range [0, 1].

In this case, $N_{\text{min}}$ and $N_{\text{max}}$ have been set respectively to 10 (also in this case, the a priori number of classes) and 200. In this set of experiments, the average recognition rate obtained on the training set has been 97.63%, whereas that obtained on the test set has been 96.6%. The negligible difference between the above rates is a sign of the good generalization power of the system. Note that system performance strongly depends on the description method adopted. The features we used are not the best possible ones, as our aim was not that of realizing a high performance recognition system of handwritten characters, but that of testing a prototyping method.

In order to investigate if the implemented system is subject to overfitting phenomena, the recognition rates during a run, on training and test set, have been taken into account. In Figure 6(a) such recognition rates, evaluated every 50 generations, during a typical run are displayed. It can be observed that the recognition rate increases with the number of generations both for the training and the test set. The best recognition rates occur in both cases nearby generation 400. Moreover, the fact that the difference between the two recognition rates does not tend to increase when that on the training set reaches its maximum, demonstrates the good generalization power of our system. Figure 6(b) shows that, as the number of generations grows, the number of prototypes becomes stable at a value corresponding to the
5.4. Comparison Findings

The results obtained by our method on the NIST data have been compared with those obtained by other two classification systems: $K$-Nearest-Neighbor ($k$-NN) and a standard LVQ neural network.

The LVQ net chosen for the comparison of our results is an improved version of the standard one, called Frequency Sensitive Competitive Learning (FSCL) (40), and is often used to compare the performances of other algorithms. In Figure 7, training and test recognition rates obtained by the LVQ classifier have been plotted as a function of the total number of prototypes (i.e. neurons) used. The best rate on the test set, 96.14% has been obtained by using 800 prototypes (80 for each class).

As for the $K$-NN classifier, let us note that an unknown sample $x$ is recognized by computing the Euclidean distance between $x$ and each of the samples in the training set. Then the sample is attributed to the most widely represented class among the $k$ nearest samples in the training set. In Table 1 the results obtained by the $K$-NN classifier are reported.

In Table 2 the results obtained by the three methods are shown. The recognition rate are quite similar in the three cases, but the number of prototypes necessary for achieving such results is significantly lower for our algorithm (EA). This implies the classification time is significantly shorter.

<table>
<thead>
<tr>
<th>$K$</th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test set</td>
<td>96.69</td>
<td>96.90</td>
<td>96.82</td>
<td>96.65</td>
<td>96.17</td>
<td>95.94</td>
</tr>
</tbody>
</table>

Table 1. The obtained results of the $K$-NN classifier as a function of $K$. 

Fig. 7. Recognition rates on training and test set, obtained by the LVQ classifier, as a function of the total number of prototypes.
<table>
<thead>
<tr>
<th></th>
<th>3-NN</th>
<th>LVQ</th>
<th>EA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mean</strong></td>
<td>96.90</td>
<td>96.14</td>
<td>96.60</td>
</tr>
<tr>
<td><strong>Std</strong></td>
<td>–</td>
<td>0.10</td>
<td>0.20</td>
</tr>
<tr>
<td><strong>N_P</strong></td>
<td>10,000</td>
<td>800</td>
<td>124.7 (5.4)</td>
</tr>
</tbody>
</table>

Table 2. The results obtained by the three compared classifiers. In the first and second rows the average test rates and the standard deviation are respectively shown. In the third row the number of prototypes used is given.

6. Conclusions

A prototyping method based on the evolutionary computation paradigm has been proposed. Clusters of points in the feature space are represented by the prototypes obtained by evolving an initial population of randomly selected feature vectors. No a-priori knowledge about both the actual number of clusters and the statistical properties of the input data is requested.

The method has been tested on both synthetic data and handwritten digits from the NIST database. The results have been compared with those obtained on the same data by using two different classification methods. The effectiveness of our approach mainly lies on the good generalization power of the system and its capability to limit the number of prototypes necessary for achieving good results.

Note that the recognition rate values reported for OCR should not be considered particularly meaningful, since our aim was not to build a top classifier, but just to present an alternative prototyping method. Of course, the absolute recognition performance also depends on the chosen features and the classification strategy.
References

REFERENCES