ELSEVIER

Pattern Recognition Letters 23 (2002) 1439–1448

Pattern Recognition Letters

www.elsevier.com/locate/patrec

Character preclassification based on genetic programming

C. De Stefano [a], A. Della Cioppa [b], A. Marcelli [b,*]

[a] *Dipartimento di Automazione, Elettromagnetismo, Ingegneria dell'Informazione e Matematica Industriale, Università di Cassino, Via Marconi, 10, 03043 Cassino (FR), Italy*

[b] *Dipartimento di Ingegneria dell'Informazione ed Ingegneria Elettrica, Università di Salerno, Via Ponte don Melillo, 84084 Fisciano (SA), Italy*

Abstract

This paper presents a learning system that uses genetic programming as a tool for automatically inferring the set of classification rules to be used during a preclassification stage by a hierarchical handwritten character recognition system. Starting from a structural description of the character shape, the aim of the learning system is that of producing a set of classification rules able to capture the similarities among those shapes, independently of whether they represent characters belonging to the same class or to different ones. In particular, the paper illustrates the structure of the classification rules, the grammar used to generate them and the genetic operators devised to manipulate the set of rules, as well as the fitness function used to drive the inference process. The experimental results obtained by using a set of 10,000 digits extracted from the NIST database show that the proposed preclassification is efficient and accurate, because it provides at most 6 classes for more than 87% of the samples, and the error rate almost equals the intrinsic confusion found in the data set. © 2002 Published by Elsevier Science B.V.

Keywords: Character recognition; Preclassification; Genetic programming

1. Introduction

The recognition of handwritten characters involves identifying a correspondence between the pixels of the image representing the sample to be recognized and the abstract definitions of characters (models or prototypes). The prevalent approach to solve the problem is that of implementing a bottom–up process for extracting and combining many pieces of information (features) that are eventually used to build up the models for each class. During the classification, such models are compared with the input sample according to the adopted classification method for deciding to which class the sample belongs. Due to the extreme variability exhibited by samples produced by a large population of writers, pursuing such an approach 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. The combination of complex classification methods with a large set of prototypes has a dramatic effect on the classifier: a larger number of prototypes requires a higher discriminating power, which, in turn, requires more sophisticated methods and algorithms to perform the classification.

* Corresponding author.

0167-8655/02/$ - see front matter © 2002 Published by Elsevier Science B.V.
PII: S0167-8655(02)00104-6

For this reason, we have investigated the possibility of using a preclassification technique whose main purpose is that of reducing the number of classes to be considered when classifying an input sample. Such a reduction affects both the performance of the classifier and its computational cost. As with respect to the performance, lowering the number of classes leads to a simpler classification problem, thus reducing the risk of confusing the right class with a wrong one. The reduction of the computational cost follows immediately from the reduction of the number of prototypes, although it is counterbalanced by the cost involved by the preclassification itself. The general problem of reducing a classifier computational cost has been faced since the 70's in the framework of statistical pattern recognition (Fukunaga and Narendra, 1975) and, more recently, within shape-based methods for character recognition (Marcelli and Pavlidis, 1994; Marcelli et al., 1997). The large majority of the preclassification methods for character recognition proposed in the literature belongs to one of two categories, depending on whether they adopt a different set of features or a different classification strategy, with respect to those adopted during the classification (Mori et al., 1984).

In this paper, developed within the framework of an evolutionary approach to character recognition (De Stefano et al., 1999), we present a novel preclassification method that uses genetic programming (GP) as a tool for learning a set of classification rules (prototypes) that describe specimen belonging to one or more actual classes. To this purpose, starting from a graph-based representation of the character shape, we compute a simpler description in terms of a feature vector. Such a vector is composed of as many elements as the types of primitives and of spatial relationships among them that can be used to describe a sample, and each element of the feature vector counts the occurrence of each feature found in the sample. Consequently, a prototype is represented by a set of assertions connected by Boolean operators, each assertion specifying the constraints on the occurrence of a feature. The proposed preclassifier works in two different modes. During an off-line unsupervised training phase, an initial population of randomly generated prototypes is evolved according to the GP paradigm in order to produce the set of prototypes that achieves the maximum coverage of the training set. At the end of the training, to each prototype is associated the list of the classes covered by that prototype. In such a list are included all the classes whose samples satisfy the constraints expressed by the prototype. At run time, after the feature extraction, the feature vector is computed for a given sample and matched against the prototypes. The desired preclassification is thus obtained by assigning to the sample the list of the classes associated to the simplest matching prototype.

Let us remark that, due to the discrete nature of our feature space and to the meaning of the features, the distance between the points in this space does not necessarily reflect the shape similarity. Therefore, none of the methods based on distance evaluation, such as principal component analysis, K-means clustering and so on, can be used. An approach similar to the one followed by us has been very recently proposed (Teredesai et al., 2001). In that study, developed on the basis of a multi-level feature extraction method, GP is aimed at selecting the optimal set of features, i.e. the set of features able to maximize the separability among the classes to be recognized. Our method, on the contrary, uses GP at a coarser classification level, to capture the similarities among character shapes, independently of the classes the characters belong to.

The paper is organized as follows. Section 2 illustrates the character shape description scheme and how it is reduced to a feature vector. In Section 3 we present our approach and its implementation, while Section 4 reports the experimental results. Concluding remarks are eventually left to Section 5.

2. From character shape to feature vector

In the framework of structural methods for pattern recognition, the most common approach is based on the decomposition of an initial representation of the sample into elementary parts, each of which can be described in a simple manner. In

this way a character is described in terms of a set of parts interrelated by more or less complex links. Such a structure is then described by a sentence of a language or by a relational graph. Accordingly, the classification is performed by parsing the sentence, so as to establish its accordance with a given grammar, or by a graph matching technique, in order to decide which prototypical graph is the most similar to the sample one. In our case, we have opted for a graph-based description and the actual procedure to compute it is articulated into three main steps: skeletonization, decomposition and description (Chianese et al., 1989). During the first step, the character skeleton is computed by means of a MAT-based algorithm, while in the following one it is decomposed in parts, each one corresponding to an arc of a circle which we have selected as our primitive. Eventually, each arc found within the character is described by the following features:

- *size*, referred to the size of its bounding box;
- *span*, represented by the angle spanned by the arc;
- *direction*, represented by the oriented direction of the normal to the chord subtended by the arc.

The spatial relations among the arcs are computed with reference to arc projections along both the horizontal and vertical axis of the character bounding box. In order to further reduce the variability still present among samples belonging to the same class, the descriptions of both the arcs and the spatial relations are given in discrete form:

- *size*: small, medium, large;
- *span*: closed, medium, wide;
- *direction*: N, NE, E, SE, S, SW, W, NW;
- *relation*: over, below, to-the-right, superimposed, included.

Those descriptions are then encoded into a feature vector of 76 elements. The first 63 elements of the vector are used to count the occurrences of the different arcs that can be found within a character, the following 13 elements describe the set of possible relations among them (Cordella et al., 1995). It is worth noting that the adopted description does not specify the way the arcs are connected to each other, but only that a connection of a certain type exists. Therefore, characters whose shapes can be decomposed in terms of the same types and number of features will be encoded into the same feature vector. This could happen, for instance, in case of samples of digits '6' and '9' both described in terms of one loop and one straight segment, lying one atop of the other.

3. Learning explicit classification rules

As mentioned in the introduction, the prototypes to be used for the preclassification are given in terms of classification rules. Since classification rules may be thought of as computer programs, a natural way for introducing them into our learning system is that of adopting the GP paradigm (Koza, 1992, 1994). Such a paradigm combines genetic algorithms and programming languages in order to evolve computer programs of dynamically varying complexity (size and shape) according to a given defined behavior. According to this paradigm, populations of computer programs are evolved by using the Darwin's principle that evolution by natural selection occurs when the replicating entities in the population possess the *heritability* characteristic and are subject to *genetic variation* and *struggle to survive*.

Typically, GP starts with an initial population of randomly generated programs composed of functionals and terminals especially tailored to deal with the problem at hand. The performance of each program in the population is measured by means of a fitness function, whose form also depends on the problem. After the fitness of each program has been evaluated, a new population is generated by selection, recombination and mutation of the current programs, and replaces the old one. Then, the whole process is repeated until a termination criterion is satisfied.

In order to implement such a paradigm, the following steps have to be executed:

- definition of the structures to be evolved;
- choice of the fitness function;

- choice of the selection method and definition of the genetic operators.

3.1. Structure definition

The implementation requires a program generator, providing syntactically correct programs, and an interpreter, for executing them.

The program generator is based on a grammar written for S-expressions. A grammar $\mathscr{G}$ is a quadruple $\mathscr{G} = (\mathscr{T}, \mathscr{V}, S, \mathscr{P})$, where $\mathscr{T}$ and $\mathscr{V}$ are disjoint finite alphabets. $\mathscr{T}$ is the *terminal alphabet*, $\mathscr{V}$ is the *non-terminal alphabet*, S is the *start symbol*, and $\mathscr{P}$ is the set of *production rules* used to define the strings belonging to the language, usually written as $v \to w$ where v is a string on $(\mathscr{T} \cup \mathscr{V})$ containing at least one non-terminal symbol, and w is an element of $(\mathscr{T} \cup \mathscr{V})^*$. For the problem at hand, the set of terminals is the following:

$$\mathscr{T} = \{a_1, a_2, \ldots, a_{76}, 0, 1, \ldots, 9, (,), \wedge, \vee, \sim, <, \leqslant, =, >, \geqslant\},$$

and the set $\mathscr{V}$ is composed as follows:

$$\mathscr{V} = \{A, X, I, M, C, B\},$$

where a_i is a variable atom denoting the ith element in the feature vector, and the digits $0, 1, \ldots, 9$ are constant atoms used to represent the value of each element in the feature vector. It should be noted that the above sets satisfy the requirements of closure and sufficiency (Koza, 1992). The adopted set of production rules is given in Table 1.

Each prototype in the initial population is generated starting with the symbol S that, according to the above grammar, can be replaced only by the symbol A (rule 1). The symbol A can be replaced by any recursive combination of logical expressions whose arguments are the occurrences of the elements in the feature vector. Therefore, each prototype corresponds to a binary tree where the leaves correspond to IMX clauses, while nodes correspond to Boolean operators. The data structure used by GP to encode the tree is a string. An example of a prototype and its internal representation is shown in Fig. 1. It is worth noting that, in order to avoid the generation of very long individuals, the probability of selecting the clause IMX is higher than the probability of selecting any other clause that appears in the second production rule listed in Table 1.

Finally, the interpreter is implemented by an automaton that computes Boolean functions, i.e. an acceptor. Such an automaton computes the truth value of the rules in the population with respect to the samples.

Table 1
The grammar for the random rules generator

Production rule no.	Production rule	Probability
1	$S \to A$	1.0
2	$A \to CBC \mid (CBC) \mid (IMX)$	0.25|0.25|0.5
3	$I \to a_1 \mid \cdots \mid a_{76}$	Uniform
4	$X \to 0 \mid 1 \mid \cdots \mid 9$	Uniform
5	$M \to < \mid \leqslant \mid = \mid \geqslant \mid >$	Uniform
6	$C \to A \mid \sim A$	Uniform
7	$B \to \vee \mid \wedge$	Uniform

3.2. Fitness function

The next step to accomplish is the definition of a fitness function to measure the performance of the prototypes. To this purpose, it should be noted

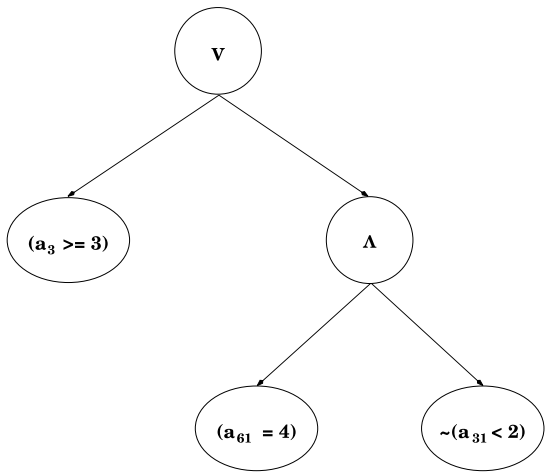

Fig. 1. An example of a prototype: the tree and the corresponding string.

that, as already mentioned in the Introduction, we are looking for a set of prototypes that, all together, achieve the highest covering rate on the training set.

In order to allow the evolution to produce a set of different partial solutions (niches) rather than a single one, as it happens with canonical evolutionary algorithms, some kind of niching must be incorporated at different levels into the fitness function (Deb and Goldberg, 1989; Mahfoud, 1995; Booker et al., 1989; Forrest et al., 1993; Horn et al., 1994). In our case, we have adopted a niching mechanism based on *resource sharing* (Forrest et al., 1993; Horn et al., 1994). According to resource sharing the cooperation and competition among the niches is obtained as follows: for each sample s_i of the training set a subset P of prototypes from the current population is selected, and each prototype in the subset P is matched against the sample s_i. In our case, a prototype p matches a sample *s if and only if p covers s*, i.e. the values of the elements of the feature vector representing the sample satisfy the constraints expressed in the prototype. Note that, according to such a definition of covering, a feature not mentioned in the prototype cannot be present in any sample covered by that prototype. If there is only one prototype matching the sample, it receives a payoff. In case that two or more prototypes satisfy the match, the prototypes whose genotype is the shortest one is selected as winner and receives the payoff. In the case of a tie, the winner is randomly selected among the deserving prototypes. At the end of the cycle, the fitness of each prototype $\phi(p)$ is computed by adding all payoffs earned:

$$\phi(p) = \sum_{i=1}^{m} c \cdot \mu(p, s_i)$$

where m is the number of samples in the training set, $\mu(p, s_i)$ is a function that returns 1 if p is the winner for the sample s_i and 0 otherwise, and c is the payoff. The criterion of selecting as winner the prototype whose genotype is the shortest one takes into account two main aspects: (i) the purpose of prototyping is that of capturing the distinctive features while neglecting the irrelevant ones; it is obvious that between two (or more) prototypes covering a given sample, selecting the simplest one ensures a higher level of distinctiveness; (ii) the prototypes are inferred during an unsupervised training whose purpose is that of maximizing the coverage of the training set; therefore, longer prototypes have higher probability of earning payoffs. As a consequence, if not counterbalanced, the learning would produce very long prototypes covering many samples belonging to many different classes, thus lacking in distinctiveness. These two aspects recall the Occam's razor principle of simplicity closely related to Kolmogorov Complexity definition (Li and Vitányi, 1993; Conte et al., 1997), i.e. "*if there are alternative explanations for a phenomenon, then, all other things being equal, we should select the simplest one*".

3.3. Selection and genetic operators

The selection mechanism is responsible for choosing among the prototypes in the current population the ones that undergo genetic manipulation for producing the new population. In this study we have used the Stochastic Universal Sampling mechanism, in that it helps the maintenance of the discovered niches (Baker, 1987).

As regards the mutation operator, we perform both *micro-* and *macro-mutation*. The micro-mutation is applied whenever the symbol selected for mutation is any terminal but a bracket, and it is responsible for changing the type of the feature, its occurrence and its constraints, as well as the Boolean operators. Therefore, it resembles closely the classical point-mutation operator.

In order to show how the micro-mutation works, let us consider the following string:

$$(a_3 \geqslant 3) \wedge (\sim (\boldsymbol{a}_{10} < 2) \vee (a_{53} \geqslant 1))$$

and suppose that the symbol selected is a_{10}. Such a symbol is replaced by the non-terminal symbol I thus leading to the string:

$$(a_3 \geqslant 3) \wedge (\sim (\boldsymbol{I} < 2) \vee (a_{53} \geqslant 1))$$

This string is then manipulated according to the grammar to obtain the new string (in such a process the probability of selecting the original symbol has been set to zero):

$$I \rightarrow a_{44} \Rightarrow (a_3 \geqslant 3) \wedge (\sim (\boldsymbol{a}_{44} < 2) \vee (a_{53} \geqslant 1))$$

The same action is performed if the symbol selected is a digit, a relation operator or one of the Boolean operators $\wedge$ and $\vee$, and it replaces the symbol with one of the non-terminal symbols X, M or B, respectively. Finally, if the selected symbol is the $\sim$ operator, it is simply deleted.

The macro-mutation is activated when the symbol to be mutated is a bracket. In this case, the relative subtree is selected and one of the following mutually exclusive operations is performed:

- *substitution*: performed with probability p_s, substitutes the selected subtree with another one randomly generated according to the grammar described in Section 3.1,
- *insertion*: performed with probability p_i, allows a new randomly generated subtree to be inserted in the selected one. To be accomplished, it requires the selection of a Boolean operator ($\wedge$ or $\vee$) for connecting a node of the selected subtree and the newly generated one.
- *deletion*: performed with probability p_d, deletes the selected subtree. It should be noted that usually insertions and deletions are mutually chosen with the same probability.

Obviously, such mutations are implemented in a way that ensures the syntactic correctness of the newly generated prototypes. It follows from the above that the macro-mutation is responsible for modifying the structure of the decision tree corresponding to each prototype in the same general way as that implemented by the classical tree-crossover operator. For this reason this operator is not used here.

To illustrate how the macro-mutation operator works, let us consider the string

$$((a_3 < 3) \wedge ((a_{61} = 4) \vee \sim (a_{31} < 2))) \vee (a_{53} \geqslant 1)$$

and suppose that the symbol selected for mutation is the first bold bracket and that the operation selected is the substitution of the relative subtree. Such a subtree is then replaced by the non-terminal symbol A, thus leading to the string:

$$((a_3 < 3) \wedge \boldsymbol{A}) \vee (a_{53} \geqslant 1)$$

Finally, such a string is modified according to the grammar as follows:

$$A \rightarrow (a_{10} > 2) \Rightarrow ((a_3 < 3) \wedge (a_{10} > 2)) \vee (a_{53} \geqslant 1)$$

If the operation selected is the insertion, the macro-mutation could generate the following string:

$$((a_3 < 3) \wedge (\boldsymbol{A} \vee ((a_{61} = 4) \vee \sim (a_{31} < 2)))) \vee (a_{53} \geqslant 1)$$

and, according to the grammar, the resulting string may be the following:

$$A \rightarrow (a_{10} > 2) \Rightarrow ((a_3 < 3) \wedge ((a_{10} > 2) \vee ((a_{61} = 4) \vee \sim (a_{31} < 2)))) \vee (a_{53} \geqslant 1)$$

Finally, if the operation selected is the deletion, the macro-mutation generates the following string:

$$(a_3 <) \vee (a_{53} \geqslant 1)$$

4. Experimental results

The ability of GP to generate classification rules in very complex cases, like the one at hand, and the preclassifier performance in terms of both efficiency and accuracy have been evaluated through a large set of experiments. The experiments were performed on a data set of 10,000 digits extracted from the NIST database and equally distributed among the 10 classes. This data set was randomly subdivided into a training and a test set, both including 5,000 samples. Each character was decomposed, described and eventually coded into a feature vector of 76 elements, as reported in Section 2.

It must be noted that the performance of a GP-based system is heavily influenced by the values of some parameters. We have divided this set of parameters into external parameters, that mainly affect the performance of the learning, and internal parameters, that are responsible for the effectiveness and efficiency of the search. As external parameters we have assumed the population size N, the number of generations G and the maximum depth D of the trees representing the rules in the population. The internal parameters are the mu-

tation probability p_m (for both micro- and macro-mutation), the substitution probability p_s, the insertion probability p_i, and the deletion probability p_d for macro-mutation, the number of rules n competing for environmental resources in the resource sharing, and the maximum depth d of the subtrees generated by the macro-mutation operator. The values for these parameters have been set experimentally, and the results reported in the sequel have been obtained with $N = 1000$, $G = 350$, $D = 10$, $p_m = 0.6$, $p_s = 0.8$, $p_i = p_d = 0.2$, $n = N$ and $d = 3$. We recall that the distributions used for generating the random strings, both in the initial population and in the mutation, are those reported in Table 1.

As mentioned before, the first experiment was aimed at evaluating the capability of the GP to deal with complex cases such as the one at hand. For this purpose, we have monitored the covering rate $\mathscr{C}$, i.e. the percentage of the samples belonging to the training set covered by the set of prototypes produced by the system during the learning. Let us recall now that, according to the ultimate aim of our work, the purpose of the learning is that of inferring prototypes able to capture the shape similarity among characters described by the adopted set of features, independently of the actual classes they belong to. Therefore, the most natural way to achieve the goal has been that of implementing an unsupervised learning mechanism, by providing the system with the feature vectors of the samples in the training set, and allowing the system to evolve the set of prototypes for achieving the highest covering rate. To this aim we have decided to stop the learning when either a covering rate equal to 100% has been reached, or when the maximum number of generations have been performed. Moreover, we have performed 10 runs in order to reduce the randomness introduced by the generation of the initial population.

The best recognition rate was 91.38%, while the worst was 83.08%. The recognition rate averaged over all the 10 runs was equal to 88.93%. In the best case, there were only 431 samples in the training set for which the system was unable to generate or to maintain a suitable set of prototypes with the time limit of 350 generations. The experimental results reported in the following will refer to the set of prototypes obtained in the best run.

A first experiment to evaluate the robustness of such a set of prototypes to shape variations was performed by matching the samples in the test set against those prototypes. Only 440 samples were not matched while all the others were correctly classified, yielding a recognition rate of 91.20%. This result is very meaningful in that we observe a loss of only 0.18% with respect to the recognition rate obtained on the training set.

A second experiment was carried out to evaluate the performance of the preclassifier. To achieve this, we preliminarily labeled the prototypes obtained at the end of the learning phase. The labeling was such that each time a prototype matched a sample of the training set, the label of that sample was added to the list of labels for that prototype. At the end of the labeling, thus, each prototype had a list of labels of the classes it covered, as well as the number of samples matched in each class.

A detailed analysis of these lists showed that most of the prototypes covered many samples belonging to very few classes and very few samples belonging to many different classes. This was due to "confusing" samples, i.e. samples belonging to different classes but having the same feature vector, thus being indistinguishable for the system. In the majority of the cases, the "confusing" samples occur very frequently in some classes, and therefore represent a "typical" shape for those classes, while they are very infrequent in other classes, therefore representing "atypical" (distorted or noisy) instances for those classes. To reduce this effect, the labels corresponding to the classes whose number of samples covered by the prototype was smaller than a given percentage π of the total number of samples covered by that prototype were removed from the list. Finally, a set of classification experiments on the test set was performed for different values of π, yielding to the results reported in Table 2. Each row in Table 2 reports the recognition rate R and the percentage of recognition rate ascribable to prototypes whose list contains 1 or 2 (1,2), 3 or 4 (3,4), 5 or 6 (5,6), or more (>6) classes. Eventually, the last column reports the error rate E, i.e. the percentage of samples

Table 2
The experimental results obtained on the test set

π	R	$(1,2)$	$(3,4)$	$(5,6)$	(>6)	E
0	0.912	0.187	0.147	0.180	0.486	0.000
1	0.908	0.210	0.234	0.182	0.374	0.004
2	0.898	0.301	0.214	0.252	0.233	0.014
3	0.887	0.340	0.219	0.329	0.112	0.025
4	0.873	0.350	0.307	0.335	0.008	0.039
5	0.860	0.394	0.325	0.273	0.008	0.052
6	0.851	0.411	0.365	0.216	0.008	0.061
7	0.838	0.445	0.401	0.154	0.000	0.074
8	0.812	0.515	0.420	0.065	0.000	0.100
9	0.800	0.579	0.377	0.044	0.000	0.112
10	0.780	0.704	0.285	0.011	0.000	0.132

covered by prototypes that do not contain the proper class in their lists. To emphasize the efficiency of the preclassifier, Fig. 2 shows the plots of the recognition rates as function of the threshold π.

The data reported in Table 2 confirm the tradeoff between the accuracy (as measured by the error rate) and the efficiency (as measured by the number of possible classes provided for a sample) of the preclassifier. The highest accuracy is reached when $\pi = 0$, since there are no errors, but the efficiency of the preclassifier is very low, because 48.6% of the recognized samples are matched by prototypes with more than 6 classes in their lists. As far as π becomes larger, the efficiency of the preclassifier improves at the expense of the accuracy: when passing from $\pi = 0$ to 4, the number of

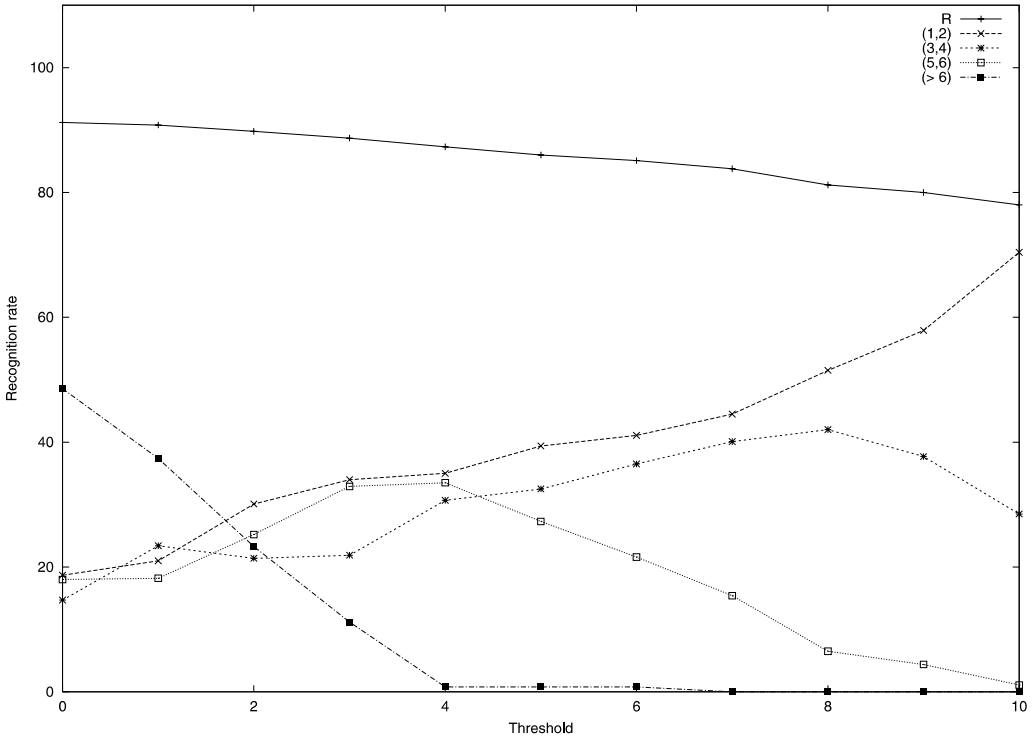

Fig. 2. The recognition rates as function of the threshold π.

samples matched by prototypes with more than 6 classes in their lists decreases from 48.6% to 0.8%, but the error rate reaches 3.9%. If we assume as measure of the efficiency of the preclassifier the percentage of recognized samples covered by prototypes whose lists contain at most 6 classes (i.e. the preclassifier is efficient if it can screen out at least 40% of the classes), the data in Table 2 show that, roughly speaking, there is a 1:12 ratio between the loss in accuracy and the gain in efficiency. They also show that when π passes from 0 to 4, the percentage of samples not actually preclassified (column 6) plus the error rate (column 7) steadily decreases from 48.6% to 4.7%. On the contrary, for $\pi > 4$ the efficiency gain a further 0.8%, but the error rate increases from 3.9% to 13.2%. Therefore, the measure obtained adding columns 6 and 7 steadily increases from 6% to 13.2%. In other words, as it is even more evident in Fig. 2, the value $\pi = 4$ represents the best compromise between accuracy and efficiency.

5. Conclusions

In this paper we have presented a novel preclassification method that uses GP as a tool for learning a set of prototypes able to capture the similarities among character shapes even if the characters belong to different classes. The proposed method adopts a fine-to-coarse description scheme, whose bottom level is a graph-based representation of the character shape, while the top level is a feature vector composed of as many elements as the types of arcs and of spatial relationships that can be used to describe a character shape. Since only the feature vector is used for the preclassification, a prototype is represented by a set of assertions, connected by Boolean operators, each one specifying the constraints on the occurrence of a feature. The prototypes are obtained by means of an unsupervised learning implemented according to the genetic programming paradigm and eventually labeled with the classes to which the samples covered by that prototype belong.

The experimental results reported in the previous section allow for two concluding remarks. The first one is that GP is very appealing in developing hierarchical handwritten character recognition systems, because it may produce prototypes at different level of abstraction, depending on the way the system is trained. The results of the first experiment show that the system was not able to achieve a covering rate of 100%, but this can be explained by noticing that the feature vectors representing the uncovered 431 samples corresponded to "isolated" points in the feature space, and therefore the niching mechanism was unable to find enough resource to populate and maintain the corresponding niches. An 100% covering rate could be achieved by increasing the population size and the number of generations, but such an increase will, in turn, result in a higher computational cost of the learning. On the other hand, the system in its current configuration already exploits all the information available in the training set, since, as it has been reported in the previous section, the covering rate remains practically unchanged when measured on the test set.

The second remark is that the learning system developed by us, although it does not receive information on which nodes are connected by which arcs in the original graph, therefore making use of only a fraction of the information carried by the character shape, is highly efficient and accurate. Since the classification is performed by matching the unknown sample against the whole prototype set to determine the winner, reducing the number of classes reduces the number of prototype to consider, thus resulting in an overall reduction of the classification time. It is also accurate, because the highest efficiency is achieved with an error rate of 13.22%, that almost equals the amount of "confusing" samples of the training set, i.e. samples belonging to at least 5 different classes but represented by the same feature vectors.

References

Baker, J.E., 1987. Reducing bias and inefficiency in the selection algorithm. In: Grefenstette, J.J. (Ed.), Genetic algorithms and their applications: Proc. Second Int. Conf. on Genetic Algorithms. Lawrence Erlbaum Association, Hillsdale, NJ, pp. 14–21.

Booker, L.B., Goldberg, D.E., Holland, J.H., 1989. Classifier systems and genetic algorithms. Artificial Intell. 40, 235–282.

Chianese, A., Cordella, L.P., De Santo, M., Marcelli, A., Vento, M., 1989. A structural method for handprinted character recognition. Lecture Notes Comput. Sci. 399, 289–302.

Conte, M., Trautteur, G., De Falco, I., Della Cioppa, A., Tarantino, E., 1997. Genetic programming estimates of Kolmogorov complexity. In: Bäck, T. (Ed.), Proc. Seventh Int. Conf. on Genetic Algorithms. Morgan Kaufmann, San Francisco, CA, pp. 743–750.

Cordella, L.P., De Stefano, C., Vento, M., 1995. Neural network classifier for OCR using structural descriptions. Machine Vis. Appl. 8 (5), 336–342.

Deb, K., Goldberg, D.E., 1989. An investigation of niche and species formation in genetic function optimization. In: Schaffer, J.D. (Ed.), Proc. Third Int. Conf. on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA, pp. 42–50.

De Stefano, C., Della Cioppa, A., Marcelli, A., 1999. Handwritten numerals recognition by means of evolutionary algorithms, Proc. 5th Int. Conf. on Document Analysis and Recognition, Bangalore, India, pp. 804–807.

Forrest, S., Javornik, B., Smith, R.E., Perelson, A.S., 1993. Using genetic algorithms to explore pattern recognition in the immune system. Evolution. Computat. 1 (3), 191–211.

Fukunaga, K., Narendra, P.M., 1975. A branch and bound algorithm for computing k-nearest neighbors. IEEE Trans. Comput. C 24 (7), 750–753.

Horn, J., Goldberg, D.E., Deb, K., 1994. Implicit niching in a learning classifier system: Nature's way. Evolution. Computat. 2 (1), 37–66.

Koza, J.R., 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA.

Koza, J.R., 1994. Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press, Cambridge, MA.

Li, M., Vitányi, P., 1993. In: An Introduction to Kolmogorov Complexity and Its Applications, Text and Monographs in Computer Science. Springer-Verlag, Berlin.

Mahfoud, S.W., 1995. Populations size and genetic drift in fitness sharing. In: Whitley, L.D., Vose, M.D. (Eds.), Foundations of Genetic Algorithms 3. Morgan Kaufmann, San Francisco, CA, pp. 185–223.

Marcelli, A., Likhareva, N., Pavlidis, T., 1997. Structural indexing for character recognition. Comput. Vis. Image Understanding: CVIU 66 (3), 330–346.

Marcelli, A., Pavlidis, T., 1994. Using projections for preclassification of character shape, in: Vincent, L., Pavlidis, T., (Eds.), Proc. SPIE Conf.—Document Recognition, vol. 2181. Los Angeles CA, 1994. pp. 4–13.

Mori, S., Yamamoto, K., Yasuda, M., 1984. Research on machine recognition of handprinted characters. IEEE Trans. Pattern Anal. Machine Intell. 6, 386–405.

Teredesai, A., Park, J., Govindaraju, V., 2001. Active handwritten character recognition using genetic programming. In: Miller, J.F., Tomassini, M., Lanzi, P.L., Ryan, C., Tettamanzi, A.G.B., Langdon, W.B. (Eds.), Lecture Notes Comput. Sci. 2048, pp. 371–379.