Evolving Binary Classifiers Through Parallel Computation of Multiple Fitness Cases

Stefano Cagnoni, *Senior Member, IEEE*, Federico Bergenti, Monica Mordonini, and Giovanni Adorni

***Abstract*—This paper describes two versions of a novel approach to developing binary classifiers, based on two evolutionary computation paradigms: cellular programming and genetic programming. Such an approach achieves high computation efficiency both during evolution and at runtime. Evolution speed is optimized by allowing multiple solutions to be computed in parallel. Runtime performance is optimized explicitly using parallel computation in the case of cellular programming or implicitly taking advantage of the intrinsic parallelism of bitwise operators on standard sequential architectures in the case of genetic programming.**

The approach was tested on a digit recognition problem and compared with a reference classifier.

***Index Terms*—Cellular programming, genetic programming, multiple classifiers, pattern recognition.**

I. Introduction

Evolutionary computation (EC) comprises a wide range of optimization techniques, inspired by the laws of natural evolution, which can be effectively applied to machine learning and pattern recognition, as well as to many other fields.

Given a task that can be mapped onto an optimization problem, EC techniques use different encoding schemes (genotypes) to explore different spaces within which solutions (phenotypes) are searched. For example, in Genetic Algorithms (GAs), the phenotype most often represents the set of parameters for a function whose structure has been formerly defined, while in Genetic Programming (GP) [1], [2], the phenotype of each individual in the population is a program. Several different GP paradigms have been described, the most commonly used being the one originally proposed by Koza [1], where programs are encoded as syntactic trees. This kind of evolutionary paradigm is computationally very intensive. First, the higher computation requirements with respect, for example, to GAs are due to the much wider search space it explores, which can become infinite, if tree depth is not limited. Second, tree-encoded solutions make decoding, crossover, and mutation much more central processing unit (CPU)-demanding than binary string-encoded solutions used in GAs. Therefore, there is great interest in developing GP implementations that improve the computational efficiency of evolution [3], [4].

At the same time, the increasing need for high-performance programs for real-time applications has fostered the development of EC paradigms that use encoding and representation schemes that make execution of evolved programs very efficient. This has led to methods that imply some degree of parallelism, both in executing the evolutionary algorithm and in the structure of the resulting programs. In this paper, we focus on two evolutionary techniques that rely, explicitly or implicitly, on parallel computation paradigms: Cellular Programming (CP) [5] and sub-machine-code GP (SmcGP) [6], [7]. We describe the use of these two evolutionary techniques within the same framework to show the advantages offered by the basic idea they share (concurrently computing multiple fitness cases), rather than to compare them with each other. We used such techniques to design two different solutions to a low-resolution character classification problem, tracing different viable paths toward maximization of classification accuracy and computation efficiency.

After briefly introducing CP and SmcGP in Section II Section III describes the basic idea underlying this work and the characterizing features of two evolutionary systems that were derived from it. Sections IV and V describe the experimental setup and results of the digit-recognition systems that were produced to test the approach. Finally, Section VI comments on the results obtained in the digit-recognition experiment and draws some conclusions on the effectiveness of the two evolutionary techniques in evolving binary classifiers.

Manuscript received August 3, 2003; revised April 25, 2004. This work was supported in part by the Italian Space Agency (ASI) under the grant "Hybrid Vision System for Long Range Rovering" and by the Italian Ministry of Education, University, and Research (MIUR) under the "Evolutionary Computation Methods for Pattern Recognition and Classification" under Grant FIL 2001. This paper was recommended by Associate Editor Bir Bhanu.

S. Cagnoni, F. Bergenti, and M. Mordonini are with the Dipartimento di Ingegneria dell'Informazione, University of Parma, Parma, Italy (e-mail: cagnoni@ce.unipr.it; bergenti@ce.unipr.it; monica@ce.unipr.it).

G. Adorni is with the Dipartimento di Informatica, Sistemistica e Telematica, University of Genoa, Genoa, Italy (e-mail: adorni@unige.it).

Digital Object Identifier 10.1109/TSMCB.2005.846671

II. Background

A. Cellular Programming

Cellular Programming [5] is an EC technique in which cellular automata (CA) [8] are evolved. In particular, CP evolves nonuniform CAs, i.e., CAs in which a different local computation is allowed for every cell. This property distinguishes nonuniform CAs from uniform CAs in which the same computation is performed throughout the whole grid of cells.

A CA has the same spatial organization as its input, with each cell taking the value of the corresponding input component as initial state. In each step, a new state is computed for each cell. The output of the state-transition function associated with each cell (i.e., the new state of the cell) depends on the present states of the cell and of the neighboring ones within an arbitrarily shaped N-cell neighborhood. The CA is typically run for a number of steps large enough to allow information to propagate across the grid in order to also let the final state of each cell be affected by the initial state of the cell farthest from it. Each cell's state-transition function can be encoded as a 2^N-bit look-up table: Each bit corresponds to the future state that the cell will reach from each of the 2^N possible configurations of the present state of the corresponding N-cell neighborhood.

Nonuniform CAs have been used in many fields, including fast hardware implementation [9] and development of associative memories [10]. The requirement that a different rule be designed for each cell in the grid dramatically enlarges the rule space and makes designing such CAs a very complex task. This is the reason why automatic design procedures, such as EC, are the methods of choice for nonuniform CA design.

In CP, the look-up table, which is evolved to implement the state-transition function for each cell, can be considered, in evolutionary terms, to be the cell's genotype. As happens with CA computation, in which the automaton state is computed based on local interactions of cells with the neighboring ones, evolution in CP is based on local interactions among the genotypes of neighboring cells. Evolution takes place according to the CP algorithm described in detail in [5], [11], whose core is summarized in pseudo-code in Fig. 1. This algorithm results in the co-evolution of a set of rules driven by a local fitness function.

As discussed above, the transition functions of the CA can be represented as bit strings. Therefore, the same mutation and crossover operators used in classical bit string-based GAs can also be used in CP.

```
for each cell i do in parallel

compute nfi (i)  // number of fitter neighbors
  if nfi (i) = 0 then
    rule i is left unchanged

  else if nfi (i) = 1 then
    replace rule i with the fitter neighbouring
    rule, followed by mutation

  else if nfi (i) = 2 then
    replace rule i with the crossover of
    the two fitter neighboring rules,
    followed by mutation

  else if nfi (i) > 2 then
     replace rule i with the crossover of two
     randomly chosen fitter neighboring rules,
     followed by mutation

end if
end parallel for
```

Fig. 1. Pseudo-code of the CP algorithm (from [5] and [11]).

It is worth pointing out that a CA having S cells with a binary state may be considered to be a processor yielding S different 1-bit outputs in response to an S-bit input. It is also worth pointing out that programs evolved by CP need to be run on special hardware [12] to get the most out of them in terms of computation speed. They can be only simulated, rather inefficiently, on sequential CPUs. This reason induced us to also explore and develop an evolutionary approach to designing binary classifiers based on sub-machine-code Genetic Programming.

B. Sub-Machine-Code Genetic Programming

Sub-machine-code GP exploits the intrinsic parallelism of bitwise instructions of sequential CPUs and can be run effectively on traditional computer architectures. Inside a sequential K-bit CPU, each bitwise operation on integers is performed by concurrently activating K logic gates of the same kind. Therefore, applying a sequence of bitwise logical operators to a K-bit integer is equivalent to executing the same program on K 1-bit operands in parallel. SmcGP is a form of GP that evolves functions in which bitwise operators are applied to packed representations of 1-bit data.

At runtime, programs evolved by SmcGP are intrinsically parallel and use operators that can be directly translated into machine code. Both properties allow programs evolved by SmcGP to be computationally very efficient.

During evolution, SmcGP efficiency is also very high. The closure requirement of tree-like genetic programs requires that any GP-evolved function that operates on a certain input data type produce an output of the same type. Therefore, in using GP to optimize a binary function $f : \{0,1\}^K \rightarrow \{0,1\}$, each phenotype actually produces a K-bit output. Such an output can be seen to be generated by a set of K (non-independent) functions $f_i, i = 1,\ldots,K$, for which fitness can be independently evaluated, as further described in Section III.

SmcGP can be applied effectively to domains in which the same operations must be performed concurrently on blocks of binary data that can be packed, for instance, into a long integer variable. This is the case for binary-pattern recognition or binary-image processing problems, in which two-dimensional patterns can be processed line-wise or block-wise.

III. Efficient Evolution of Binary Classifiers Using CP and SmcGP

SmcGP and CP can be powerful tools for developing high-performance binary classifiers in terms of both accuracy and computation speed. They can also be used in many more general cases; in fact, any M-class classifier of arbitrary complexity can be implemented as an ensemble of $C >= M$ specialized binary classifiers. Different possible architectures are possible for such an ensemble. They generally yield better results than a single, equivalent classifier, thanks to richer and/or redundant processing of information.

However, the choice to use a multiclassifier approach [13], [14] must be supported by methods that produce fast accurate classifiers very efficiently, for several reasons. First, since accuracy of an ensemble of classifiers depends on the error rates of the single classifiers, developing ensembles of classifiers require that each component be very accurate. Second, developing an ensemble of binary classifiers is usually more time consuming than developing a single, equivalent, classifier. Finally, running an ensemble of classifiers is generally computationally demanding.

We propose a general framework within which the two evolutionary approaches under consideration can be used to design binary classifiers that meet the above-mentioned requirements. The performance of binary classifiers thus obtained has been assessed on a low-resolution digit recognition problem.

A common architecture has been used for the binary classifiers in both approaches. Following a typical detection scheme, each classifier is associated with one class and is required to output 1 when the input pattern belongs to the corresponding class, and 0 otherwise. To solve M-class problems, a set of M such binary classifiers can be used; the final classification can be derived from the response of all classifiers. This is, for instance, the typical architecture and training strategy used in M-class classifiers based on feed-forward neural networks. However, while, in neural networks, paths leading from the input to the output layer share several weights and classifiers are usually trained concurrently, in the approaches described in this paper, classifiers are evolved independently of one another.

The ideal situation for an ensemble of M classifiers occurs when only the classifier corresponding to the class of the input pattern outputs 1, whereas all other outputs are 0. As pointed out in [14], where this classification architecture is described with particular reference to the use of GP-evolved classifiers, there are, quite intuitively, two trivial cases in which this strategy fails. One occurs when no classifier produces a high output, whereas the other one occurs when more than one classifier produce one as output. The problem of taking a final decision in the latter case can be tackled with several different approaches, such as a hierarchy of increasingly specialized classifiers or an *a posteriori* statistical approach.

When a hierarchy of classifiers is considered, a second layer of "tie-breaker" classifiers (one for every possible couple of conflicting classes) is developed, and their output is interpreted according to criteria similar to those used in sports tournaments. The output of each classifier is seen as the result of a match between the two classes under consideration, and the decision can be based, for example, upon a knock-out or a round-robin tournament mechanism. In another, more computationally expensive but usually more performing approach, for each training pattern I_i, all outputs $\{O_i\}$, $i = 1, \ldots, C$ of the basic classifier set are recorded, then packed into a single pattern $P_i = \{O_1, \ldots, O_C\}$, and finally classified by a so-called "stacked generalizer" [15]. The stacked generalizer operates as a look-up table, labeling each pattern P_i as belonging to the class with the highest *a posteriori* probability to produce P_i.

Therefore, from the point of view of classification strategy, the general architecture of the evolutionary classifiers considered in this paper is quite conventional. The peculiar features of the approach are the high degree of parallelism in each classifier and the criterion by which the output of each classifier is computed. Regarding the capability to perform parallel computation, CP derives it directly from the intrinsic properties of the cellular automata paradigm on which it is based; instead, SmcGP derives it implicitly from the use of bitwise operators applied to packed representations of arrays of 1-bit data.

If D is the size in bits of the input space, both CP and SmcGP compute a function $f : \{0,1\}^D \to \{0,1\}^S$, $S > 1$.

Such a function can be seen as a set of S alternative binary-output functions $f_i : \{0,1\}^D \to \{0,1\}$, $i = 1, \ldots, S$, which are computed concurrently in each function evaluation. The output space size S is equal to the size of the pattern to be classified in CP and to the size of the word into which 1-bit data are packed in SmcGP.

Therefore, decoding each genotype I_k requires two steps. In the first step, the S-bit output function directly represented by the genotype is decoded. In the second one, S fitness evaluations are made by iteratively considering each of the S output bits obtained in the first step as the 1-bit outputs of S different classifiers. This yields S different solutions f_{ki}, $i = 1, \ldots, S$ to the problem at hand. As long as a fitness function F is defined for the problem, a different fitness value $F(f_{ki})$ can therefore be associated with each solution. The fitness $F(I_k)$ of each individual (in the SmcGP approach) or of the whole automaton (in the CP approach) is considered to be the maximum fitness obtained in the second step.

$$F(I_k) = \max_i (F(f_{ki})), \quad i = 1, \ldots, S. \tag{1}$$

This definition of fitness implies that the best-performing output bit of each individual/automaton is taken as its actual 1-bit output.

Besides being beneficial from the point of view of computation efficiency, this strategy favors the extraction of the most relevant features of the patterns under classification. This occurs because, in both approaches, there exists a direct morphologic and geometric correspondence between input and output. In CP, the output pattern represents the final state that a CA, whose initial state has been set equal to the input pattern, reaches after performing local computation. Therefore, it is often possible to identify the region of the input pattern by which the final state of the cell that is taken as classifier output is most influenced. This also happens in SmcGP when the whole input pattern can be packed in one word. However, SmcGP often needs to span a larger and more complex input space, evolving functions that operate on different "slices" of the input pattern. Nonetheless, operations performed on those slices are local as well.

These properties suggest that the region of the input pattern by which the output value of the highest-fitness bit is most influenced contain the most significant feature for the class under consideration.

IV. EC-Based Recognition of License Plate Characters

To show the potentials of the approaches we propose, we report results obtained using CP and SmcGP to develop a set of binary classifiers for license plate-digit recognition. Fig. 2 shows some examples of the digits that were used in the experiment to give an idea of their average quality.

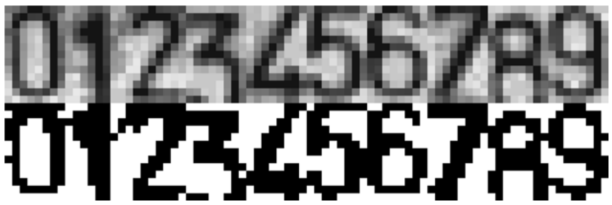
Fig. 2. Some digits extracted from the license-plate database used in the experiments (above) and their binarized counterparts (below).

In this section, we briefly introduce the application and the most relevant implementation details of the two approaches. More detailed information in those regards can be found in [16]–[18].

At first glance, character classification for license plate-recognition [17], [19] seems to have much in common with traditional Optical Character Recognition (OCR). However, in OCR, characters are classified after text has been scanned at high resolution. In license-plate recognition, instead, classifiers deal with very low-resolution patterns that are usually obtained from low-quality snapshots altered by optical distortions and perspective effects. Therefore, classification of license-plate characters is usually a much more critical task with respect to character recognition in printed documents.

Former applications of GP to character recognition mostly deal with character preprocessing or feature extraction from printed [20] or hand-written characters [21], [22]. In [23], genetic programming has been used to evolve modular neural networks that classify handwritten digits, showing significant performance improvements with respect to both standard multilayer perceptrons and nongenetically optimized modular networks.

The main focus of our work is on the method and not on the application. We have used character recognition as a benchmark mainly because of the availability of a rather large data set on which we had previously tested other techniques; in particular, feedforward and learning vector quantization (LVQ) neural nets. Results obtained with such techniques have been used as a reference in evaluating the results of our approach.

In applying CP [16] and SmcGP [18] to low-resolution license-plate character classification, the data set we used included binary patterns of size 13×8 pixels, representing digits from 0 to 9, taken from a database collected by Società Autostrade SpA (Italian highway company) at highway toll booths and trivially binarized using a threshold of 0.5 on a scale ranging from 0 to 1. The data set includes 11 034 patterns almost uniformly distributed among the ten classes under consideration. The training set includes 6024 patterns, whereas the test set contains 5010.

A. Cellular Programming-Based Approach

In the CP-based experiment, the CA consists of a two-dimensional grid of the same size as the patterns to be recognized (13×8), as shown in Fig. 3. The state-transition function of each cell is based on local computation performed on a cross-shaped neighborhood[1]. To classifiy a pattern, the initial state of the CA is set equal to the pattern, which is then considered to be the CA input. Then, each classifier is run for 24 steps, and the state reached is considered to be the output of the CA. According to the neighborhood definition, at time $t + 1$, the state of each cell is influenced by the state of all adjacent cells at time t, at time $t + 2$ by the state of all adjacent cells, and by the state of all cells

[1]This is the so-called Von Neumann neighborhood, which is defined, in terms of offset from the cell under consideration, as $V_N = \{(0,0), (-1,0), (+1,0), (0,-1), (0,+1)\}$

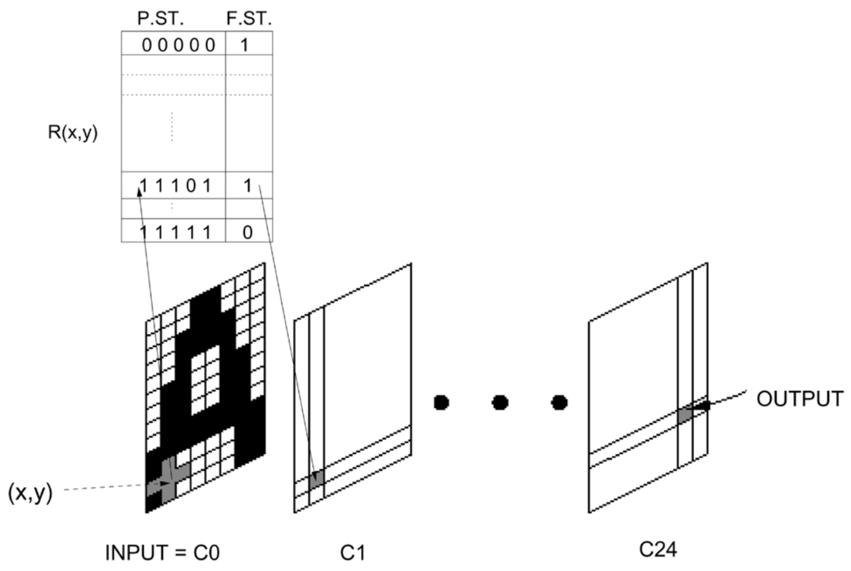

Fig. 3. CP-based approach. The initial state C0 of the automaton is equal to the input pattern. After 24 steps, the classifier output corresponds to the state of the highest fitness cell in C24. The neighborhood considered in updating the state of cell (x, y) in C0, and the corresponding cell in C1 is highlighted in grey. An example of rule table $R(x, y)$ associated with cell (x, y) is also shown: The first column reports the present state (P.ST.) of the five bits in the neighborhood of (x, y); the second column reports the future state (F.ST.) of (x, y).

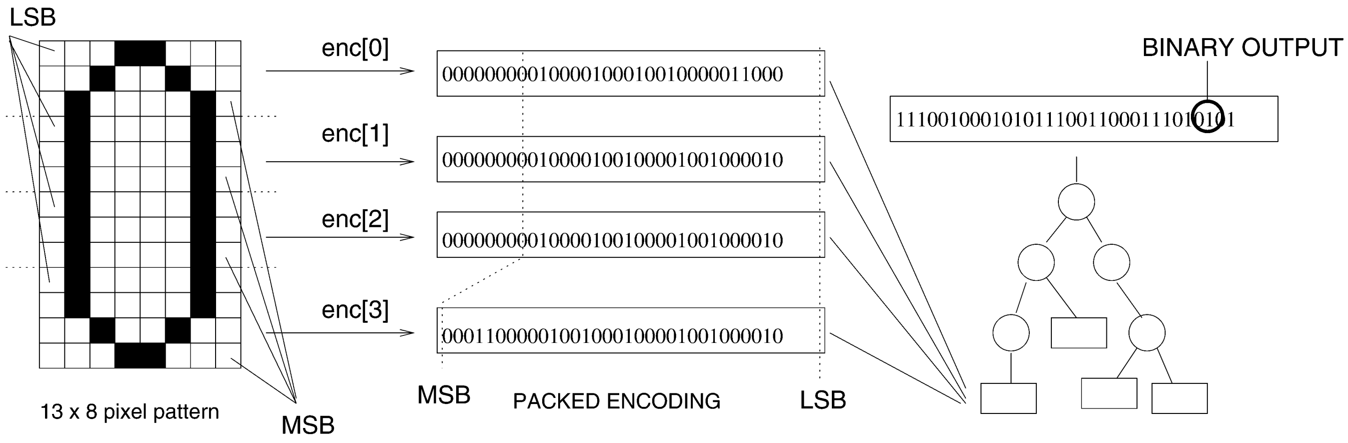

Fig. 4. SmcGP approach. Each character is encoded as four 32-bit integers, which are then processed by the GP-evolved function. The value of the highest fitness bit within the 32-bit output of the function is the classifier output.

adjacent to them, and so on. It can therefore be shown that, in our case, 24 steps are more than enough to allow the input to one of the cells that are located in opposite corners to influence the output of the other one.

In CP, the set of local state-transition functions that define the global state-transition function of the nonuniform CA can be seen as a set of evolving single-individual populations that are devoted to solving different but reciprocally influencing problems, among which genetic matter can be exchanged. Considering the highest fitness cell as the output for the classifier lets evolution occur as the result of competition, among neighboring cells, to reach the highest fitness. However, at the same time, cells also cooperate to produce the desired output.

For each binary classifier described in Section V, the evolutionary algorithm was run five times, for 20 000 generations, with a mutation rate of 0.001.

B. SmcGP-Based Approach

To apply SmcGP to character classification, each 13×8 two-dimensional binary pattern is encoded row-wise as four 32-bit words. Since the number of bits in each pattern (104) is not a multiple of 32, the 24 least significant bits of the first three words encode the first nine rows of the pattern, whereas the whole fourth integer is used to encode the last four rows, as shown in Fig. 4. Preserving the same pixel order in each of the 32-bit components of the code is the only specification for the encoding. The encoding would have been equally effective if, for example, the pixel had been packed in reverse order most significant bit to least significant bit (MSB to LSB) or column-wise.

The function set (see Table I) is composed by the main bitwise Boolean operators and by a set of circular shift operators SHxn, in which the 32-bit word LSB is considered to be adjacent to the MSB. Shift operators are characterized by different shift direction [$x = L$ (left) or R (right)] and entity ($n = 1$, 2, or 4 bits). The use of bitwise operators yields parallel computation; however, whatever the combination of such operators, the value of the kth output bit depends only on the value of the kth input bit. The addition of the shift operators makes the computation power of the functions evolved by SmcGP similar to that achievable by a CA.

The terminal set comprises the four unsigned long integers into which the input pattern is encoded and an unsigned long integer ephemeral random constant (ERC) [1], [2], [24], which can take values within the whole range spanned by 32-bit unsigned integers.

TABLE I
FUNCTION AND TERMINAL SETS USED TO EVOLVE THE CLASSIFIERS WITH SMCGP

Function set			Terminal set		
Functional	Arity	Notes	Terminal	Type	Notes
AND	2	bitwise AND	pat[0]	term	rows 1-3
OR	2	bitwise OR	pat[1]	term	rows 4-6
NOT	1	bitwise NOT	pat[2]	term	rows 7-9
XOR	2	bitwise XOR	pat[3]	term	rows 10-13
SHxn	1	$x \in \{L, R\}, n \in \{1, 2, 4\}$	R1	ERC	unsigned long

TABLE II
PERFORMANCES OF THE BASIC CP-BASED SET OF DIGIT CLASSIFIERS

Digit	Sensitivity	Specificity
0	0.988	0.990
1	0.971	0.966
2	0.905	0.983
3	0.915	0.973
4	0.988	0.962
5	0.947	0.977
6	0.938	0.923
7	0.946	0.989
8	0.958	0.850
9	0.951	0.913

We developed our own code to implement CP, whereas the SmcGP classifiers were evolved using *lil-gp1.01* [24], which is a popular package that implements tree-based GP. Ten GP runs were performed to evolve the classifier associated with each class, adding up to 100 runs overall. In each run, a population of 1000 classifiers was evolved and randomly initialized with trees having 4 as the minimum and 7 as the maximum tree depth. A crossover rate of 80%, a mutation rate of 2%, and a reproduction rate of 18% were used during evolution. The strategy chosen for selection was tournament selection with the tournament size equal to 7. Each run was stopped after 1000 generations. For each class, the classifier that best performed on the training set was considered to be the result of the algorithm.

C. Fitness Functions

As previously noticed, for each evaluation of an individual I_k (or of a CA), a number $S > 1$ of binary-output functions, corresponding to different solutions $f_{ki}, i = 1, \dots, S$, are actually computed. For each f_{ki}, the corresponding fitness function $F(f_{ki})$ is computed.

In our experiments, we used two fitness functions that are semantically very similar but lead to behaviors that differ significantly; one is better suited to evolving redundant multiclassifier architectures, whereas the other one is better suited to evolving single-layer mutually exclusive sets of classifiers.

The first fitness function is defined as the root mean square of sensitivity and specificity:

$$F1(f_{ki}) = \sqrt{\frac{(\text{Sensitivity}_i{}^2 + \text{Specificity}_i{}^2)}{2}}$$

where Sensitivity_i is computed on the whole training set as the frequency with which the ith output equals 1 when a pattern belonging to the target class is presented as input; Specificity_i is similarly computed as the frequency with which the ith output equals 0 when a pattern that does not belong to the target class is presented as input. This fitness function produces classifiers characterized by good balance between specificity and sensitivity. This is usually the best choice if a robust criterion is available for resolving possible ambiguous situations in which more than one class-specific binary classifier produce a high output. This fitness function, despite scaling values differently from the more popular $\text{Sensitivity}_i \times \text{Specificity}_i$ induces exactly the same ordering as the latter, to which it is totally equivalent when a rank-based selection scheme, such as tournament selection, is used.

If, instead, the goal is to minimize ambiguities, better results can be achieved using the following fitness function:

$$F2(f_{ki}) = 1 - \sqrt{\frac{FP_{ki}^2 + FN_{ki}^2}{N_p^2 + N_n^2}}$$

where FP_{ki} is the number of false positives generated by f_{ki} in classifying the training set, FN_{ki} is the the number of false negatives, N_p is the number of positive examples in the training set, and N_n is the number of negative examples. This fitness function induces the evolution of high-specificity classifiers, with very high positive predictivity and a very limited number of ambiguous classifications. This can be easily explained by considering that given an N-class classification problem with a roughly uniform distribution of patterns in the training set, negative cases occur about $N - 1$ times as often as positive cases. Both terms of the fitness function are normalized using the same constant, and the function favors situations in which the number of false positives, besides being minimized, is close to the number of false negatives. Therefore, specificity, which is inversely proportional to the number of false negatives, is greatly favored.

V. RESULTS

We performed two sets of experiments (one using CP and one using SmcGP) aimed at evaluating the basic properties of the two approaches to parallel evolution of binary classifiers and at experimenting with possible multiclassifier extensions. All results we report were obtained on the 5010-character test set, unless otherwise specified.

In the first set of experiments, which are aimed at evolving an ensemble of redundant classifiers, we used CP with fitness function $F1$.

We first evolved a basic set of ten class-specific binary classifiers. Table II reports the results obtained by each of the ten binary classifiers in terms of sensitivity and specificity.

Along with the basic one, we evolved a second set of classifiers, in which each classifier distinguishes between two target classes. The output of each of these classifiers must be 1 if the input pattern belongs to the first class and 0 if the input pattern belongs to the other one; in the case where the input pattern belongs to any other class, either result is acceptable. The simpler tasks to which these classifiers are devoted makes their performance very high, as shown in Table III. As

TABLE III
PERFORMANCES OF THE PAIRWISE CLASSIFIERS. C_1 AND C_2 ARE THE CLASS OF THE FIRST AND SECOND DIGIT, RESPECTIVELY

Digit Pair	$P(1\|C_1)$	$P(0\|C_2)$	Digit Pair	$P(1\|C_1)$	$P(0\|C_2)$	Digit Pair	$P(1\|C_1)$	$P(0\|C_2)$
01	0.997	0.992	18	0.997	0.995	45	0.995	0.995
02	1.000	0.993	19	0.992	0.993	46	0.978	0.987
03	1.000	0.991	23	0.990	0.991	47	0.990	0.994
04	1.000	0.983	24	0.985	0.983	48	0.988	0.972
05	1.000	0.990	25	0.983	1.000	49	0.975	0.993
06	0.994	0.992	26	0.988	1.000	56	0.939	0.979
07	0.994	0.997	27	0.985	0.952	57	0.990	0.997
08	0.982	0.998	28	0.988	1.000	58	0.959	0.988
09	0.997	0.993	29	0.985	0.995	59	0.985	0.978
12	1.000	0.980	34	0.987	1.000	67	0.989	0.992
13	1.000	0.979	35	0.979	0.983	68	0.984	1.000
14	0.976	0.995	36	0.985	0.992	69	0.989	0.983
15	0.990	0.978	37	0.998	0.997	78	0.994	0.972
16	0.997	0.992	38	0.989	0.988	79	0.994	0.990
17	0.982	0.983	39	0.987	0.981	89	0.991	0.983

anticipated in the introduction, a set of such classifiers makes it possible to develop several different multiple-classifier architectures.

In testing a few such architectures, the best results were achieved using a round-robin approach, in which a majority-vote decision was taken by considering the outputs of the $M-1$ classifiers in which each class is involved. Such an approach yielded 96.4% global accuracy.

Better results were achieved by the approach based on the stacked generalizer, which could reach 97% global accuracy.

The drawback of this solution is that the number of classifiers $(M \times (M-1)/2)$ that need to be designed and, therefore, the size of the pattern space of the stacked generalizer, depend quadratically on the number of classes. For ten classes, 45 classifiers are needed, which makes such an approach still acceptable for building digit classifiers. For a 26-class problem, as would be in alphabetic character classification, 325 classifiers would be needed; a full implementation of this approach would therefore be very heavy in terms of evolution time. However, the confusion matrix[2] of a classifier is typically rather sparse, and few specific misclassifications are usually much more likely to occur than all others. This may dramatically limit the number of pairwise classifiers that actually need to be designed to achieve good performances. For example, the data in Table II suggest that most misclassifications and/or ambiguous outputs for the basic classifier involve the digits 8 and 9. However, this is compensated for by the performance of the corresponding pairwise classifier, which is rather good, as reported in Table III.

In the second experiment, in which we used SmcGP, we aimed at evolving a basic set of ten classifiers that maximized the probability of producing nonambiguous classifications, i.e., to produce one high and nine low outputs in response to as many input patterns as possible.

Fitness function $F2^*$, which is semantically equivalent to $F2$, was chosen to control evolution since it privileges specificity over sensitivity, while maximizing both.

$$F2^*(f_{ki}) = 1 - \sqrt{\frac{FP_{ki}^2 + FN_{ki}^2 + K_S * \text{size}_k}{N_p^2 + N_n^2}}.$$

[2]The confusion matrix of a classifier C is a matrix in which each element $m_{i,j}$ represents, percentually, how many times a pattern of class i has been classified by C, as belonging to class j.

In this slightly modified version of $F2$, the term size_k is the number of nodes in the tree-like encoding of f_k. The constant K_S is such that only a fractional part, which is proportional to size_k, is added to the numerator (which is always an integer). This allows a smaller classifier to have better fitness with respect to a larger one of equal accuracy. It also prevents trees from growing beyond reasonable sizes without needing to explicitly impose a limit to tree depth.

Evolution yielded trees of size varying from less than 100 up to over 700 nodes. Depending on the resulting classifier size, each of the 100 runs of SmcGP required from 4 to 33 hr, with an average of about 9 hr, on the computer used for the experiments: a 2.8-GHz Pentium-IV PC running under Linux 2.4.

The tree-like classifiers evolved using SmcGP were tested after being converted from prefix notation, which is typical of syntactic trees, to infix notation, and finally translated into C code for compilation. Table IV reports specificity and sensitivity of the ten classifiers, along with data related with their complexity and computation efficiency. In particular, the table reports the number of nodes in the tree-like representation produced by *lil-gp* and the processing time in microseconds required by running each classifier after compiling it using the *gcc* 2.95 C compiler with all optimizations on.

As can be noticed, specificity of SmcGP classifiers is always above 99.4%. Processing time varies from 0.04 to 0.164 μs and is not exactly proportional to classifier size since basic bitwise Boolean functions can be virtually run in one clock tick, whereas circular shift operators are at least three times as demanding since they are composed by two basic shifts followed by one OR operation.

The basic classifier set could reach a nonambiguous output configuration (only one classifier produced one as output) for 94.83% of the patterns in the test set, with a classification accuracy of 98.56%.

In evaluating the results, the modified Learning Vector Quantization (OSLVQ) [25], [26] character classifier embedded in the APACHE license plate-recognition system [17] was used as reference for comparison. In experiments performed while developing APACHE, such a network (whose size is automatically adjusted during learning) performed much better than several feedforward neural net architectures trained with the backpropagation algorithm for which the best results were never better than 97.7%.

TABLE IV
PERFORMANCES OF THE SmcGP-BASED CLASSIFIERS. TIMES ARE COMPUTED AS AVERAGE RUNTIME IN TEN EXECUTIONS OF THE CLASSIFIER ON THE TEST SET, AS RECORDED BY $gprof$ (GNU PROFILER)

	0	1	2	3	4	5	6	7	8	9
Specificity	99.87	99.56	99.69	99.45	99.58	99.42	99.69	99.69	99.47	99.58
Sensitivity	97.21	98.00	95.21	95.81	95.21	93.81	94.41	94.01	93.21	96.01
Tree nodes	61	86	167	284	109	211	213	86	328	212
Processing time (μs/PIV2.8GHz)	0.041	0.048	0.092	0.160	0.080	0.118	0.147	0.070	0.164	0.123

TABLE V
PERFORMANCE OF THE REFERENCE LVQ CLASSIFIER ON THE TEST SET

	0	1	2	3	4	5	6	7	8	9
Specificity	99.80	99.97	99.93	99.91	99.87	99.82	99.71	99.76	99.82	99.93
Sensitivity	99.00	99.00	99.00	98.00	99.00	98.20	99.00	99.40	98.80	97.41
Global Accuracy	98.68									

Table V reports the performance of the reference LVQ classifier in the same form, with data derived from the confusion matrix of the classifier.

In the ambiguous cases, we directly used the LVQ reference classifier as a "tie-breaker." This was necessary only in a limited number of cases (5.17%), which kept efficiency high, since an LVQ classifier is much more computationally demanding than an SmcGP-based one. The average processing time for a single pattern classification required by the set of SmcGP-evolved classifiers, computed over 10 000 runs of the classifier on the whole test set, is 1.05 μs. This is about 60 times as fast as the LVQ reference classifier, which requires 63.42 μs to perform the same task on the same PC.

The global accuracy of such an SmcGP + LVQ classifier was 98.18% versus 98.68% achieved by the LVQ classifier. This means that a significant improvement in computation efficiency could be achieved using SmcGP, limiting the loss of accuracy to 0.5%, which is a figure that, despite being relevant, keeps accuracy well above the best results we had previously obtained using feedforward neural networks.

VI. FINAL REMARKS

A novel approach to the design of (ensembles of) binary classifiers has been described and implemented using two evolutionary techniques (CP and SmcGP) that rely on parallel computation. The main advantage offered by such an approach is the combination of good accuracy with high computation efficiency both during evolution and at runtime.

The paper has offered examples of binary classifiers designed using different fitness functions that allow designers to privilege accuracy while keeping computation load more than reasonably low, or to aim at efficiency, without sacrificing performance too much.

In particular, results obtained with a single set of SmcGP classifiers in recognizing two-dimensional binary patterns were particularly relevant. The performance was close to that yielded by the reference LVQ classifier, while computation efficiency was increased by a factor of about 60, thanks to the implicit parallelism of the single classifiers, and to the simplicity of the global architecture.

We observed, however, that results obtained in our systematic approach based on ten 1000-generation evolution sessions for each classifier on the test set were slightly worse (even if not significantly) than results obtained in a set of preliminary experiments, whereas results obtained on the training set were generally better. This difference could be explained with the higher number of generations through which the evolutionary algorithm was run in the final experiments (in the preliminary experiments only few times had evolution been run for 1000 generations). Longer evolution might have caused the classifiers to over-fit data in the training set. A possible solution to this problem could be increasing the value of K_S in $F2^*$ to let evolution be driven by accuracy until a sufficiently high value is reached and by tree size in the last generations. Some late experiments seem to confirm this hypothesis.

When no CA-oriented hardware on which CP and CP-evolved CAs can run is available, PC-simulated cellular automata run much slower than classifiers obtained with SmcGP, which makes CP generally less appealing than SmcGP. However, CA-oriented hardware, besides performing actual parallel computation, provides large grids in comparison with the size of the patterns under consideration. All single or pairwise classifiers could then be possibly accommodated on a grid at one time and be run concurrently, with an appropriate spatial duplication of the input patterns. In such a case, a dramatic speed enhancement could be obtained as the implementation could fully take advantage of parallelism featured by the CA computation paradigm.

In both cases, when parallel computation can be fully exploited, the approach proposed in this paper makes it possible to evolve functions that can be used in hard real-time applications. If classification of small patterns is involved, such as in license-plate recognition, the actual gain in computation time for the whole application may be almost neglectable, even with such classification efficiency as achieved by SmcGP. However, in applications in which larger binary patterns or images need to be processed, and/or the binary function evolved needs to be used as a convolutional filter, the impact of such techniques may be dramatic. The almost one-to-one correspondence between the instruction set used and the corresponding translation into machine code made possible by SmcGP, the power of CA architectures offered by CP, jointly with concurrent evaluation of multiple fitness cases during evolution, make the approach proposed in this paper a powerful tool for automatic design of real-time pattern-recognition or image-processing systems.

REFERENCES

[1] J. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.

[2] W. Banzhaf, F. Francone, J. Keller, and P. Nordin, *Genetic Programming: An Introduction*. San Francisco, CA: Morgan Kaufmann, 1998.

[3] D. Andre and J. Koza, "A parallel implementation of genetic programming that achieves super-linear performance," *Inform. Sci.*, vol. 106, no. 3–4, pp. 201–18, 1998.

[4] M. Keijzer, "Alternatives in subtree caching for genetic programming," in *Proc. Seventh Eur. Conf. Genetic Programming*, vol. 3003, M. Keijzer, U.-M. O'Reilly, S. M. Lucas, E. Costa, and T. Soule, Eds., Coimbra, Portugal, Apr. 5–7, 2004, pp. 328–337.

[5] M. Sipper, *Evolution of Parallel Cellular Machines: The Cellular Programming Approach*. Berlin, Germany: Springer-Verlag, 1997.

[6] R. Poli and W. B. Langdon, "Sub-machine-code genetic programming," in *Advances in Genetic Programming 3*, L. Spector, W. B. Langdon, U. M. O'Reilly, and P. J. Angeline, Eds. Cambridge, MA: MIT Press, 1999, ch. 13, pp. 301–323.

[7] R. Poli, "Sub-machine-code GP: New results and extensions," in *Proc. Second Eur. Workshop Genetic Programming*, vol. 1598, W. B. L. R. Poli, P. Nordin, and T. Fogarty, Eds., 1999, pp. 65–82.

[8] S. Wolfram, *Theory and Applications of Cellular Automata*, Singapore: World Scientific, 1986.

[9] P. P. Chaudhuri, D. R. Chowdhury, S. Nandi, and S. Chattopadhyay, *Additive Cellular Automata: Theory and Applications*. Los Alamitos, CA: IEEE Comput. Soc. Press, 1997.

[10] M. Chady and R. Poli, "Evolution of Cellular-Automaton-Based Associative Memories," Univ. Birmingham, Sch. Comput. Sci., Birmingham, U.K, Tech. Rep. CSRP-97-15, 1997.

[11] M. Tomassini and E. Sanchez, *Toward Evolvable Hardware*. Berlin, Germany: Springer-Verlag, 1996.

[12] T. Toffoli and N. Margolus, *Cellular Automata Machines: A New Environment for Modeling*. Cambridge, MA: MIT Press, 1987.

[13] J. Kittler, "Combining classifiers: A theoretical framework," *Pattern Anal. Applicat.*, vol. 1, no. 1, pp. 18–27, 1998.

[14] J. K. Kishore, L. M. Patnaik, V. Mani, and V. K. Agrawal, "Application of genetic programming for multicategory pattern classification," *IEEE Trans. Evol. Comput.*, vol. 4, no. 3, pp. 242–258, Sep. 2000.

[15] D. H. Wolpert, "Stacked generalization," *Neural Netw.*, vol. 5, pp. 241–259, 1992.

[16] G. Adorni, F. Bergenti, and S. Cagnoni, "A cellular-programming approach to pattern classification," in *Proc. First Eur. Workshop Genetic Programming*, vol. 1391, LNCS, W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty, Eds., Berlin-Heidelberg, Germany, Apr. 1998, pp. 142–150.

[17] G. Adorni, F. Bergenti, S. Cagnoni, and M. Mordonini, "License-plate recognition for restricted-access area control systems," in *Multimedia Video-Based Surveillance Systems: Requirements, Issues and Solutions*, G. L. Foresti, P. Mähönen, and C. S. Regazzoni, Eds. Boston, MA: Kluwer, 2000, pp. 260–271.

[18] G. Adorni, S. Cagnoni, M. Gori, and M. Mordonini, "Efficient low-resolution character recognition using sub-machine code genetic programming," in *Soft Computing Applications*. ser. Advances in Soft Computing, A. Bonarini, F. Masulli, and G. Pasi, Eds. Berlin, Germany: Physica-Verlag, 2003, pp. 35–46.

[19] J. A. G. Nijhuis, M. H. ter Brugge, K. A. Helmolt, J. P. W. Pluim, L. Spaanenburg, R. S. Venema, and M. A. Westenberg, "Car license plate recognition with neural networks and fuzzy logic," in *Proc. IEEE Int. Conf. Neural Networks*, vol. 5, 1995, pp. 2232–2236.

[20] D. Andre, "Learning and upgrading rules for an OCR system using genetic programming," in *Proc. IEEE World Congr. Comput. Intell.*, vol. 1, 1994, pp. 462–467.

[21] C. D. Stefano, A. D. Cioppa, and A. Marcelli, "Character preclassification based on genetic programming," *Pattern Recogn. Lett.*, vol. 23, pp. 1439–1448, 2002.

[22] A. Lemieux, C. Gagné, and M. Parizeau, "Genetic engineering of handwriting representations," in *Proc. Eighth Int. Workshop Frontiers Handwriting Recogn.*, 2002, pp. 145–150.

[23] S. Cho and K. Shimohara, "Evolutionary learning of modular neural networks with genetic programming," *Appl. Intell.*, vol. 9, pp. 191–200, 1998.

[24] D. Zongker and B. Punch. (1996) Lil-gp 1.01 User's Manual. Michigan State Univ., East Lansing, MI. [Online]. Available: ftp://garage.cse.msu.edu/pub/GA/lilgp

[25] T. Kohonen, *Self-Organization and Associative Memory*, Second ed. Berlin, Germany: Springer-Verlag, 1988.

[26] S. Cagnoni and G. Valli, "OSLVQ: A training strategy for optimum-size learning vector quantization classifiers," in *Proc. First IEEE World Congr. Comput. Intell.*, Jun. 1994, pp. 762–765.