Fundamenta Informaticae 87 (2008) 1–20
IOS Press

A Neuro-Genetic Framework for Pattern Recognition in Complex Systems

S. Bandini, L. Vanneschi*
Complex Systems and Artificial Intelligence (C.S.A.I.) Research Center
Dept. of Informatics, Systems and Communication (D.I.S.Co.)
University of Milano-Bicocca, 20126 Milan. Italy
{bandini, vanneschi}@csai.disco.unimib.it

A. Wuensche,
Dept. of Informatics, School of Science and Technology, University of Sussex, UK
andy@ddlab.org

A. Bahgat Shehata
Complex Systems and Artificial Intelligence (C.S.A.I.) Research Center
D.I.S.Co., University of Milano-Bicocca, Italy
bahgat@csai.disco.unimib.it

Abstract. This paper presents a general framework to automatically generate rules that produce given spatial patterns in complex systems. The proposed framework integrates Genetic Algorithms with Artificial Neural Networks or Support Vector Machines. Here, it is tested on a well known 3-values, 6-neighbors, k-totalistic cellular automata rule called the "burning paper" rule. Results are encouraging and should pave the way for the use of the proposed framework in real-life complex systems models.

Keywords: Complex Systems, Cellular Automata, Pattern Recognition, Automatic Rule Generation, Machine Learning, Genetic Algorithms, Artificial Neural Networks, Support Vector Machines.

*Address for correspondence: Complex Systems and Artificial Intelligence (C.S.A.I.) Research Center, Dept. of Informatics, Systems and Communication (D.I.S.Co.), University of Milano-Bicocca, 20126 Milan. Italy

1. Introduction

The task of designing and producing Cellular Automata (CA) rules that exhibit a particular behavior is generally considered a very difficult one. Several solutions to automatically solve this problem by means of computer simulations were proposed (see for instance [37, 20, 14]). In all those works, however, the objective was to find CA rules that, for instance, performed simple computational tasks or that could be used to simulate logic gates. The objective of this work is to find rules able to generate given spatial *patterns*, or, in other words, rules that produce configurations that a human observer could consider, in some generic sense, *similar* to the ones generated by a given target rule. This task is an important one for a number of practical applications in many real-life complex systems like crowds or natural ecosystems, like for instance forests [2, 3]. In those applications, the ability of recognizing and modeling particular spatial patterns can be of help in decision making or for preventing cataclysms, like landslides or inundations. However, considering the task of measuring how similar a given configuration is to another one from the perspective of a computer system clearly has some difficulties. For instance, one might be tempted to consider a cell by cell comparison as a sort of initial measure of similarity, but this measure clearly fails if the sought for patterns appear in different locations of the grid, rotated or in different sizes. This implies that we need a less ingenuous approach to measure the similarity between two configurations, and this can be very difficult to obtain by means of a computer program. In addition to the huge size of the search space (i.e. the set of all the possible rules), another factor that contributes to make this problem hard is the fact that the behaviour of a CA rule is not easy to predict just by looking at the syntactical representation of the rule itself, and two rules with extremely similar representations can result both in almost identical or incredibly different global dynamics [45].

These factors, among the others, led us to choose *Genetic Algorithms* (GAs) [17, 15] to explore the rule space, given their implicit parallelism and their ability to search difficult and complex spaces. Potential solutions (or individuals) evolved by the GAs are CA transition rules represented as strings of characters as in [45]. The function used to express their quality (fitness function) has to measure the "similarity" of the configurations they generate with the ones generated by a given target rule. Because of the difficulties mentioned earlier and given that the concept of "similarity" is very informal and hard to define, we attempted to use the models generated by several Machine Learning approaches as kernels to calculate the fitness of each CA rule. Machine Learning techniques seemed to be particularly appropriate for the problem we were facing because they would free us from the need to design an algorithm to classify configurations, and they could possibly "learn" even a qualitative concept such as the definition of the patterns we were interested in. We focused on the Machine Learning methods that were more promising for their capacity of extracting "hidden" relations between the features of the elements to be classified. In particular, we found that *Support Vector Machines* (SVMs) [39, 11, 6] and *Artificial Neural Networks* (ANNs) trained with a *Backpropagation* learning algorithm [31] (also called *Multilayer Perceptron*) are probably among the best techniques that can be used for this kind of problem.

To the best of our knowledge, this work represents the first attempt to produce a general framework for automatically generating CA rules corresponding to particular spatial patterns by means of a set of Machine Learning strategies. One requirement of this work is that the presented framework has to be general, i.e. it does not have to depend on the kind of pattern it looks for and on the particular target rule; nevertheless we have chosen a particular rule as a prototype to test the proposed framework, and took the kind of configurations it produced as a sort of "goal". In particular, we focused on 3-values, 6-neighbours

k-totalistic CA rules defined on two-dimensional hexagonal lattices, and chose as prototype a particular rule, introduced in [46] and called the "burning paper" rule.

This paper is structured as follows: Section 2 contains a brief survey of the current state of the art. In Section 3 we present the "burning paper" rule, that we will use as a benchmark to test the proposed framework. Section 4 presents the general architecture and functioning of the proposed framework based on GAs. In Section 5 we discuss and motivate the Machine Learning methods that we have used to realize the most important component of the proposed framework, which should recognize particular spatial patterns and calculate the fitness of CA rules accordingly. Section 6 discusses some of the experimental results we have obtained on the "burning paper" rule. Finally, Section 7 concludes this work and offers some hints for future research activities.

2. Previous and Related Work

Cellular automata (CAs) are discrete dynamical systems that have been studied theoretically for years due to their architectural simplicity and the wide spectrum of behaviors they are capable of [10, 43]. CAs are capable of universal computation and their time evolution can be complex. But many CAs show simpler dynamical behaviors such as fixed points and cyclic attractors. CAs that can be said to perform a simple "computational" tasks have been studied in many contributions in the last few years. One such tasks is the so-called *majority* or *density* task in which a two-state CA is to decide whether the initial state contains more zeros than ones or *vice versa*. In spite of the apparent simplicity of the task, it is difficult for a local system as a CA as it requires a coordination among the cells. As such, it is a perfect paradigm of the phenomenon of *emergence* in complex systems. That is, the task solution is an emergent global property of a system of locally interacting agents. Indeed, it has been proved that no CA can perform the task perfectly i.e., for any possible initial binary configuration of states [23]. However, several efficient CAs for the density task have been found either by hand or by using heuristic methods, especially evolutionary computation [28, 27, 35, 1, 16, 8]. For a recent review of the work done on the problem in the last ten years see [21] and for a deep study on the complexity of the problem, see [40]. Analogous studies for another interesting collective CA problem: the synchronization task can be found in [18, 12]. The work presented in this paper is clearly related to all the previously quoted ones, since its objective is automatically generating new CA rules. The main difference between those contributions and the present one is that in this work we don't want to generate rules that have a precise behavior, but rules that have a similar behavior to a given one. For this reason, it is not easy to define the quality (or fitness) function for the candidate rules generated by an heuristic method. In particular, a measure of distance between the target attractor and the configurations generated by the candidate rules is not what we are looking for, since it would tend to generate rules with identical behaviors to the given one, disregarding for instance rules which can generate the same attractors, shifted, translated or rescaled. For this reason, we prefer to use the term "pattern" instead of attractor or fixed point.

The use of CA as an instrument to perform pattern recognition is not new. The first contributions that have appeared in this field were aimed at using CA for the recognition of formal languages. In [37], it has been shown for the first time how CA could be used to recognise context free languages, while in [20] this result was generalised even to non context-free ones. Successively, CA have been used to recognize many different kinds of patterns. A rather deep theoretical introduction to the use of CA for pattern recognition, followed by a set of examples can be found in [30]. A related but significantly different

contribution is [7] where the use of a particular kind of CA (called Probabilistic CA) is studied as an instrument to perform pattern recognition. The idea of using Machine Learning methods in conjunction with CA for recognizing complex patterns was proposed in [38] and further developed in [29] where a cellular system using Associative Neural Networks is presented. Pattern recognition in the field of image processing has received a growing interest in the last few years. A CA based framework for this kind of patterns is proposed in [25]. Although interesting, however, these results are very far from the scope of this paper, where we try to address the problem by the opposite viewpoint: we do not use CA to recognize patters, but we look for an instrument to recognise the patterns generated by CA.

A wide amount of literature about the use of GAs for evolving CA rules exists. A review can be found in [26]. The work of Sipper and coworkers represents a noteworthy contribution to the field (see for instance [32, 33, 36, 34]).

Also, the use of GAs for pattern recognition in CA is not new: in [14, 24] the use of GAs was proposed to design Multiple Attractor CA (called MACA) to perform pattern classification. The idea is that it is possible to use appropriate CA to perform classification by seeing what basin of attraction (i.e. the output) is eventually reached when starting from specific initial configurations (i.e. the input).

3. The Burning Paper rule

In this work, we will focus on a specific subset of the CA rules: the family of k-totalistic rules [4]. For this kind of transition functions the look-up table depends just on the total numbers of cells having each state value in the neighbourhood of a given cell, regardless their position. Because of this property, this kind of rules can be encoded using tables that are much smaller than the ones used with generic rules. In particular, the size L of a k-totalistic rule table is[1]

$$L = \frac{(v+k-1)!}{k!(v-1)!} \tag{1}$$

where v is the number of cell states, and k is the number of cells in the neighbourhood. Since they consider only the *number* of cells having a specific value in the neighbourhood, without considering their position, k-totalistic rules are intrinsically symmetric (or *isotropic*).

The transition rule we use as a prototype in this work was introduced by Wuensche [46], and published on his web site about the DDLab software[2]. It was called "burning paper" rule because of the peculiar patterns it generates in its dynamics (figure 1 shows an example snapshot produced by it, and the rule can be easily examined by supplying its k-code to the DDLab program).

The "burning paper" rule (expressed in table 1) is a k-totalistic rule, defined on a regular, hexagonal, two-dimensional lattice. The cardinality of the cell state set is 3 (we call 0, 1 and 2 the three possible cell states), while the neighbourhood of a cell is composed of the 6 cells whose centres lie in the directions determined by the unit vectors $v_1, v_2, \ldots v_6$ given by:

$$v_i = \left(\cos\left((2i-1)\frac{\pi}{6}\right), \sin\left((2i-1)\frac{\pi}{6}\right)\right), \quad 1 \leq i \leq 6. \tag{2}$$

[1] The formula comes from elementary combinatorics, and it is the number of combinations with repetitions, where v is the number of objects from which you can choose and k is the number to be chosen.

[2] The rule can be found at the address `http://www.cogs.susx.ac.uk/users/andywu/multi_value/dd_life.html`, under "predator-prey dynamics".

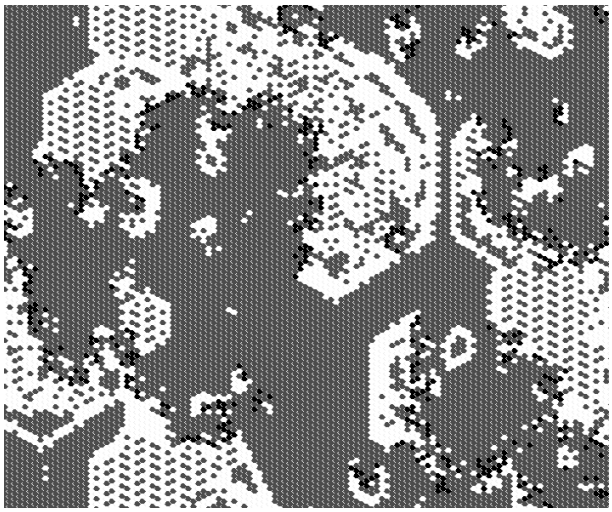

Figure 1. A configuration generated by the rule we are considering as prototype. (`k-code 120202210120111211121211101 01`)

Table 3 shows the state of each cell at the next time step as a function of the number of 0s, 1s and 2s in its neighborhood; 28 possible neighborhood configurations exist. Each one of those configurations can be assigned an index (from 0 to 27 in the leftmost column of Table 3) and the values in the rightmost column of Table 3 can be interpreted as a 28 length string indexed by those configurations. This string is often referred to as *k-code* of a rule and it completely defines a rule. The burning paper rule, identified by the k-code `120202210120111211121211101 01`, generates a specific kind of dynamics, with two "populations" whose interactions recall the ones that can be found in many prey-predator systems. In particular, one of the populations (with state `2`, white[3] in all the figures) is typically growing, while the second one (cells whose state is `0`, coloured in black) is able to survive only near white ones, and tends eventually to consume the former. In this case, and in the rest of this article, cells whose state is `1` (grey) are considered as "background".

The features we just described are the ones we are looking for in this work: we are interested in finding other k-totalistic, 3-values, 6-neighbours rules defined on hexagonal lattices that exhibit a similar behaviour. We consider a rule *interesting* if it produces configurations that a human observer can consider *similar* to the ones generated by the "burning paper" rule. In particular, we focus on rules whose dynamics is composed mostly of configurations that exhibit, for example, this couple of properties:

1. white cells tend to "stick together", i.e. they usually form more or less compact groups;

2. black cells are localized mostly on the edge of the white aggregations.

It is important to consider the previous two points as *examples* of the features we are looking for: they are not meant to completely qualify the properties of the configurations we are interested in, and are neither a formalisation nor a definition of such a vague concept as similarity.

[3]From now on we will be referring to the cell states by colours.

Index	Neighbourhood			State
	0s	1s	2s	
0	0	0	6	1
1	0	1	5	0
2	0	2	4	1
3	0	3	3	0
4	0	4	2	1
5	0	5	1	1
6	0	6	0	1
7	1	0	5	2
8	1	1	4	1
9	1	2	3	2
10	1	3	2	1
11	1	4	1	1
12	1	5	0	1
13	2	0	4	2
14	2	1	3	1
15	2	2	2	1
16	2	3	1	0
17	2	4	0	2
18	3	0	3	1
19	3	1	2	0
20	3	2	1	1
21	3	3	0	2
22	4	0	2	2
23	4	1	1	0
24	4	2	0	2
25	5	0	1	0
26	5	1	0	2
27	6	0	0	1

Table 1. The “burning paper” rule, expressed as a table. (`k-code 120202210120112112121110101`)

4. Structure and Functioning of the Proposed Framework

The general architecture of the framework we present in this paper is shown in figure 2. Its high-level structure is the same as the one of a standard GA as, for instance, described in [17, 15], including the initialization of a population of (typically random generated) potential solutions (or individuals) followed by a loop aimed at fostering individuals of better quality at each iteration (also called generations). This loop iterates four basic steps consisting in the evaluation of the fitness of all the individuals in

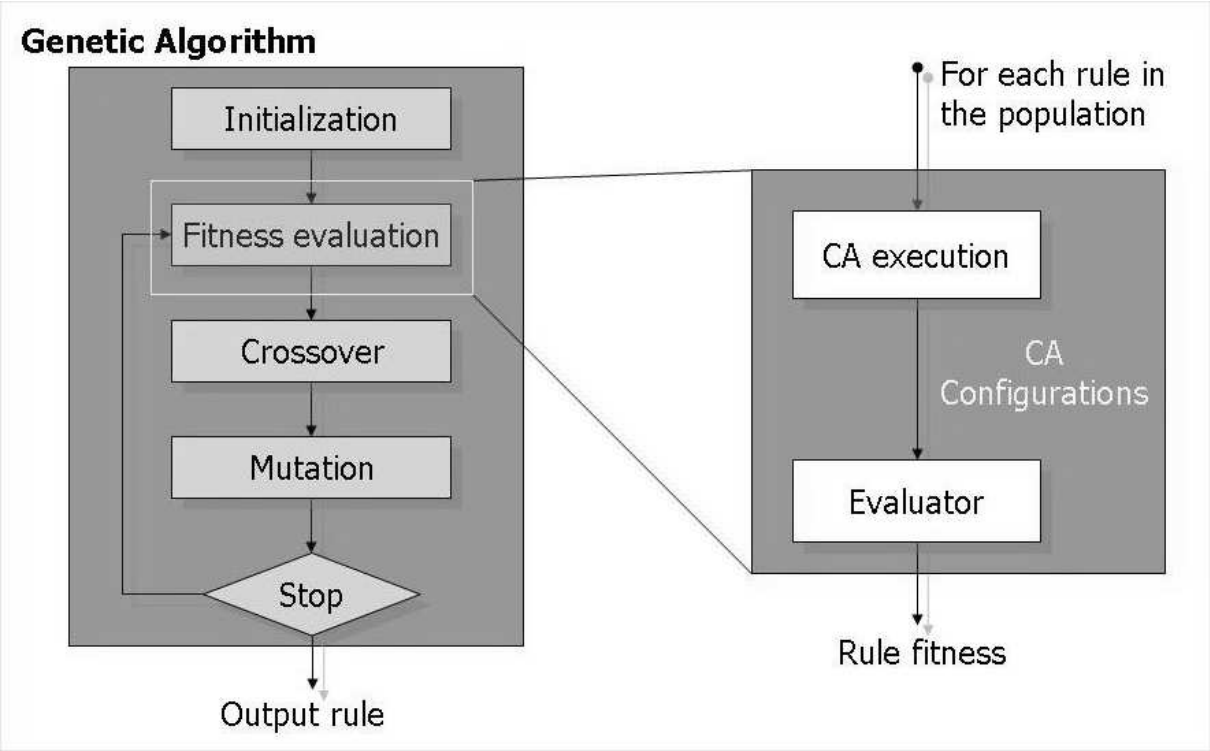

Figure 2. A graphic representation of the general architecture of the framework presented in this paper.

the population and the application of genetic operators such as selection, crossover and mutation. This generates a new population, possibly composed by individuals with better fitness. Potential solutions contained into the population are CA rules coded as strings of characters in a way that closely resembles the one used by Wolfram [42] to identify the rules of Elementary CA with binary strings. In particular, each rule can be identified by a string, called *k-code*. In our case, therefore, rules are represented as 28 characters long strings, whose alphabet is determined by the possible state values (i.e. `0`, `1` or `2`). Those individuals are then manipulated using the standard one-point variants of the crossover and mutation operators [17, 15].

The main difference between the proposed framework and a standard GA is in the fitness evaluation. To define the fitness of each GA individual, we need an instrument able to distinguish the patterns that show some *similarity* with the ones generated by the "burning paper" rule from the ones that do not. This, in addition to the difficulty of creating an efficient (even from the computational standpoint) human-made similarity function for this task, led us to choose Machine Learning. In the following Section, we will describe our analysis of two different Machine Learning approaches for this task. For now, however, we simply assume that we have generated a *model* (by means of a Machine Learning method) that is able to accomplish this task. In other words, this model is a sort of "black box" able, when given a lattice configuration, to output a number. The more the input configuration exhibits the features we are looking for, the higher the output value will be. Since we are trying to evaluate the behaviour of CA rules regardless the initial lattice state, the fitness score of a rule individual should be the result of several

runs from different, random[4] starting configurations, in order to reduce the risk of misclassification due to spurious dynamics caused by some particular choices of the initial state. This forced us to take a compromise between precision and efficiency: we chose to run the CA rule from a given number of different starting configurations, take sample snapshots from the dynamics they generated (one per run) and consider the average of the output values of the classifier model as the resulting fitness value. This led us to design the fitness function as reported in the following algorithm:

```
fitness := 0;
for s := 0 to R do
      lattice := new Lattice(W, rule);
      lattice.step(S);
      for c := 0 to C do
      begin
           lattice.step(S);
           fitness := fitness + model(lattice);
      end
endfor
fitness := fitness/(R*C);
```

where W is the width (and height) of the CA lattice, R is the number of times we run the CA on different initial states, C is the number of snapshots we take for each CA run, T is the number of steps to skip in fitness evaluation (Transient Length), S is the number of steps between subsequent snapshots and `model(lattice)` is the output of our model on `lattice`, i.e. a number equal to 1 if the current `lattice` configuration has been classified as "similar" to the target one by our model and zero otherwise. In our experiments, we have used the following values for these parameters: $W = 60$, $R = 5$, $C = 5$, $T = 30$, $S = 30$.

Each rule was executed and evaluated on $W \times W$ lattices, and then was iterated starting from R random initial configurations for a random number of steps between 0 and $S - 1$, after the mandatory execution of T steps. The execution of at minimum T steps was enforced to avoid considering the CA dynamics during the initial transient, because during that period the automaton was very likely to be heavily influenced by the specific initial states. For each run of the CA, after the initial transient, we captured C initial configurations, each separated by S steps from the following one, in order to ensure that rules are classified according to their characteristic, long-duration behaviour.

One important objection to the proposed approach could be that, by reducing the number of samples (i.e. considering only C snapshots per run, and only R runs from different initial states), the fitness function could be not very precise in measuring how "good" a rule individual is. Moreover, since the initial configurations are chosen randomly, the fitness function of an individual is neither *deterministic* nor *monotone*, even if the rule it represents is deterministic and does not change between generations. This, however, is not as true as one might expect: because of the nature of the GAs, high-fitness individuals can survive, unchanged, longer than the other ones. This means that the fitness of such individuals is evaluated at least once per generation and therefore, if a "bad" individual got a high fitness because of the particular choice of the initial states used to evaluate it, it is quite unlikely that this eventuality

[4]The starting configurations are generated by setting the state each cell to a random value in $\{0, 1, 2\}$ with uniform probability.

would happen again at the next generation. If it does, that could mean that maybe the individual we are considering is not as bad as we thought.

One of the main advantages of this way to compute the fitness value of an individual is the fact the the Machine Learning model used to classify instances can be really considered as a sort of "black box". Therefore, it can be easily replaced with different models (even human-written procedures). This feature can be useful to use the same framework we developed to test the performances of other classifiers, or even to look for rules with different behaviours.

5. The Evaluator Component

The evaluator component is the central and most important aspect of the framework presented in this paper. We used the Weka platform [41] implementation for the Machine Learning methods, since it includes many different classifiers and allows extensive testing on the models that can be obtained though it. An important point in using Weka (but typical of the Machine Learning paradigm) is the choice of the attributes (features) to use in classification. In this section, we describe how we transformed static CA lattice configurations to the instances we used to train the classifiers, pointing out the pros and cons of the choices we made. We performed a good amount of experiments but we list here only the ones that have returned the most promising results: models based on Support Vector Machines (presented in section 5.1) and on Multilayer Perceptron (presented in 5.2).

5.1. Support Vector Machines

We adopted the LIBSVM implementation of SVM by Chang and Lin [9], ported to Weka by El-Manzalawy and Honavar [13]. The SVM-based models allow larger training sets with respect to other classifiers. In addition, the instances themselves can include a huge number of attributes. For this reason, we adopted a model that was able to faithfully encode the information contained in the lattice configurations: we defined a 3-value attribute for each cell in the lattice. This results in instances with many attributes (namely $w \times h$, where w and h are, respectively, the width and the height of the lattice). In our tests, we considered 60×60 lattices, resulting in 3600 attributes that encode, quite explicitly, the spatial configuration of the lattices.

The parameters we adopted in the generation of the SVM are the defaults of the LIBSVM wrapper for Weka. They are listed in Table 2 to allow the reproducibility of the experiments (see [19] for a description of the meaning of those parameters and how they influence the performances of the classifier).

In our experiments, we created different models using training sets of growing sizes (100, 200, 500, 1000 and 2000 elements), with instances selected in order to ensure that every set is included in the larger ones. The training sets were created using the following method:

- one half of the instances (tagged as **yes**) were generated from a set of snapshots (called **Y**) of the dynamics generated by the "burning paper" rule;
- the other half (class **no**) was generated from a set of configurations (called **N**) produced by randomly chosen (but different from the one used to populate class **yes**) rules.

In both cases, each configuration was taken by iterating the application of a rule for a random number of steps (between 30 and 500, with uniform probability) on a different initial configuration, chosen

Parameter	Value	Description
type	C-SVC	
coefficient	0	
cost	1	
kernel degree	3	
ε	0.0010	*tolerance of termination criterion*
kernel type		*Radial Basis Function (see [41])*
normalise	false	
probability estimates	true	
use shrinking heuristic	true	

Table 2. Parameter setting of SVMs used in our experiments.

randomly, but with uniform distribution of state values. We did not consider configurations produced within less than 30 steps from the initial state because we wanted to allow the system to settle to its "typical" behaviour [44, 5] before starting to take snapshots. The performance of the resulting models were evaluated using both a 10-fold cross-validation on the training set and a separate test set of 600 instances not used during training. A summary of the results is presented in Table 5.1.

	M_{100}	M_{200}	M_{500}	M_{1000}	M_{2000}
cross-validation					
accuracy	96.00%	95.50%	97.00%	97.10%	97.65%
precision (**yes**)	0.926	0.933	0.943	0.947	0.956
recall (**yes**)	1	0.98	1	0.998	0.999
precision (**no**)	1	0.979	1	0.998	0.999
recall (**no**)	0.92	0.93	0.94	0.944	0.954
custom test set					
accuracy	93.32%	93.82%	94.82%	94.99%	94.99%
precision (**yes**)	0.893	0.9	0.915	0.918	0.918
recall (**yes**)	1	1	1	1	1
precision (**no**)	1	1	1	1	1
recall (**no**)	0.849	0.86	0.883	0.887	0.887

Table 3. Results of the tests on the SVM-based models. Precision and recall are listed separately for each class (**yes** and **no**).

Despite the very promising results obtained by the classifiers themselves, the SVM-based approach performed quite poorly when used in the GA algorithm. Even if the first SVM model allowed some

times the discovery of rules that could be considered, to some extent, acceptable more often it did not, resulting in different, even though interesting. This is due to the fact that the SVMs appeared to classify instances by using, as a main criterion, the *frequencies* of the cell states in the single configurations. The configurations that belong to class **yes** have all a similar and characteristic frequency distribution that can be considered, somehow, as a sort of "fingerprint" or "signature" of the rule that was used to generate them. Even if this fact helps in discarding, with a good degree of confidence, configurations that were not generated by the rule we examined earlier, it means also that the early SVM model can easily make mistakes that result in false positives. To prove that this intuition is correct, we generated a new data set using lattice configurations designed to have the same state densities of the ones used to generate class **yes**. To guarantee this property, the instances were obtained by modifying in a random way lattice configurations generated using the "burning paper" rule. In particular, the resulting configurations, that obviously had the same state frequencies of the configurations used to generate class **yes**, were shuffled until they were significantly different from the original ones. The result was a new data set (we will call it **S**) of configurations that did not show the properties we look for, but have the same global state densities of the configurations in class **yes**. Since the resulting lattice snapshots look "random" when compared to the ones in **Y**, they are all tagged as **no**. Testing revealed that each of the models whose performances are reported in Table 5.1 incorrectly classifies practically every instance of **S**.

Furthermore, this SVM-based approach has another weakness: because of the format of the instances, the models can be used (in training, testing and actual classification) only with instances obtained from configurations with the same geometry. For instance, it would be impossible to use a model obtained with a training set generated from 60×60 lattice configurations with 40×40 ones, because of the different number of attributes that will be present in the resulting instances.

5.2. Multilayer Perceptron

When Artificial Neural Networks are trained, the cardinality of the input instances can become an issue, in particular when a large number of hidden units are required to process information, as it is generally the case for Multilayer Perceptron trained with the Backpropagation learning rule. For this reason, we did not use the Multilayer Perceptron with the instances we described in the previous section: instead of having an attribute for each lattice cell, we tried to encode the *spatial* information about the configurations using the concept of neighbourhood. This choice was inspired by the fact that the previous models, after training, used density information as a major criterion for classification. This led to an hypothesis: if the number of the cells with a particular state at any given time step can help identifying a lattice configuration, counting the number of cells having a specific neighbourhood should provide even more information about the spatial properties of a configuration. Wuensche in [44] used the neighbourhood frequencies (*rule-table look-up* frequencies, or k-*block* frequencies) to calculate the input-entropy and then classify the behaviour of Cellular Automata. Therefore, we tried to use the neighbourhood counts as a set of features for classification, hoping that it would result in better classification and it would allow the perceptron to distinguish also configurations that have the same state densities, but are the product of different rules. The new instances, therefore, are now composed by $3 + 28 = 31$ attributes, the first 3 (attributes 0, 1 and 2) being the number of cells with states 0, 1 and 2, respectively, while the following 28 are calculated as follows: the value of attribute $3 + i$ is the number of cells in the current configuration whose neighbourhood is the one identified by index i in the table codying the "burning paper" rule.

Table 4 contains the parameters used to generate the Multilayer Perceptron models. As previously, we used the default values provided by Weka [41].

Parameter	Value	Description
auto build	true	add and build hidden layers
decay	false	causes a decrease in the learning rate
hidden layers	a	(attribs + classes) / 2
learning rate	0.3	
momentum	0.2	
normal to binary filter	true	
normalise attributes	true	
normalise numeric class	true	
random seed	0	
reset	true	
training time	500	*number of epochs to train through*
validation set size	0	*do not perform validation*

Table 4. Parameters used to generate the Multilayer Perceptron model.

The model was generated from a mixed training set composed of 8000 instances: 50% of them was obtained from configurations in **Y**, 25% from **N** and the remaining 25% from **S**. The instances generated from configurations in **S** were added to the training set in order to make it harder, for the classifier, to rely heavily on the first 3 attributes to discriminate between classes. The results of the first test, performed using 10-fold cross-validation, were extremely impressive: the perceptron-based model was able to correctly classify 99.26% of the instances. In order to compare this model with the ones that can be built using SVMs, we used the same training set to generate a new SVM model (the SVM parameters were still the ones in Table 2). The newly obtained model performed very poorly (its accuracy was only of 50.73%), showing that changing only the attributes to be included in the instances is not enough to obtain better results, but it is necessary to also adopt teh right classifier. Table 5.2 reports a comparison of the performances of the Multilayer Perceptron and of the SVMs.

These results are encouraging and pave the way for the use of the models generated by the Multilayer Perceptron to calculate the fitness of GAs individuals.

Furthermore, it is important to notice that the models obtained using the Multilayer Perceptron are not tightly bound to the lattice size: if the instances attributes are state and neighbourhood *frequencies*, they do not depend on the number of cells.

6. General Behavior of the Framework

Due to the nature of the task we are trying to accomplish, it is not possible to provide in this section quantitative results, i.e. to measure “how much” our experiments were successful. We can, however, list some of the rules we obtained through evolution and some of the configuration they produce.

	SVM	**multilayer perceptron**
accuracy	50.73%	99.26%
precision (**yes**)	0.504	0.99
recall (**yes**)	1	0.996
precision (**no**)	1	0.995
recall (**no**)	0.015	0.99

Table 5. Comparison between the results of the models based on Support Vector Machines and Neural Networks, calculated using 10-fold cross validation

The structure of the Genetic Algorithm we used to evolve CA rules is fairly standard: we used Tournament Selection as selection scheme, i.e. each time we needed to select an individual for mating, we chose the best one from a group of 20 (*TournamentSize* in Table 6) randomly chosen ones. The initial populations we considered were composed of 600 randomly generated rules. The other parameters used are summarized in table 6. The two probability values chosen for crossover and mutation were among the most widely used in previous works with Evolutionary Algorithms [17, 15, 22].

Parameter	**Default**	**Description**
p_c	0.95	Probability of crossover
p_m	0.001	Probability of mutation
Iterations	100	Maximum amount of iterations to perform
TournamentSize	20	The size of the tournament to use in selection
PopulationSize	600	The number of individuals in the population

Table 6. Parameters of the Genetic Algorithm

We have performed 500 indipendent GA executions. Table 7 reports 18 rules (represented by their k-code) that the GA has been able to find (we do not report all the 500 rules found in the 500 GA executions to save space).

By examining the strings that represent some of the rules obtained from experimentation it emerges that some of the rule components (i.e. characters) are more important than others in determining the nature of the behaviour of a rule. This is not a complete surprise: Wuensche and Adamatzky [47] have highlighted this aspect when considering the effect of mutations on a given rule. In other words, it is easy to imagine that, from a set of rules such as the one in Table 7, it is possible to extract a sort of *schema* [17, 15], that can distinguish all the rules that generate the kind of behaviour we are looking for. This fact confirms the suitability of the choice of using GAs as an optimization method (see for instance [17, 15] for a detailed discussion of schema theory and its importance in GAs).

Figures 3, 4, 5, 6, 7 and 8 report the snapshots of some of the configurations generated by 6 of the rules reported in table 7 (the ones marked with a "$*$" in the table). We do not report snapshot from all the 500 rules found by the 500 GA executions that we have performed to save space. Nevertheless, we can state that all the 500 rules found by the GA have generated configurations that a human being would probably consider "similar" and all of them are very "similar" to the ones shown in these figures.

	k-code
	12222121201011111122111222 11
	22222202020100211011011111 22
	12222012110021211121211101 00
	12222221112111201011111100 11
	12122222002111210011111101 01
	12122222002111210011111101 01
	12222221002111001100111122 11
	12121121012110211000111111 11
	12222122210111101101111120 10
	12122221112000111111211111 12
	12122022012110011001011101 10
	12221121102010011000111111 20
*	12121222102021110101111111 11
*	12122122122111200000211111 20
*	12122221122021210010011110 22
*	12122222200002110010111111 11
*	12222122212000201100111111 20
*	12222221022110001110111120 11

Table 7. Some rules obtained using the search approach we describe here. The codes marked with a "∗" are the ones displayed in the remainder of this section.

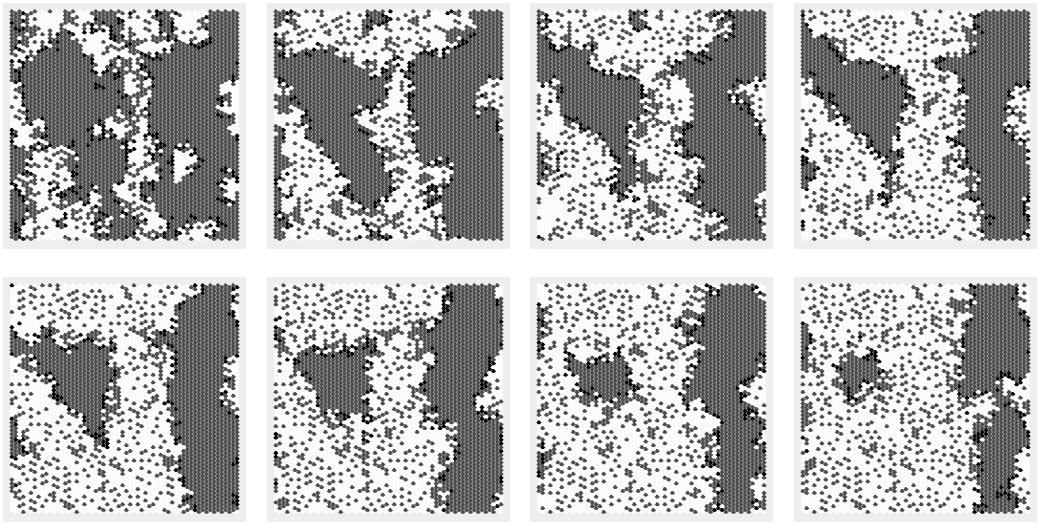

Figure 3. Snapshots of the configurations produced by rule 12121222102021110101111111111 at iterations 15, 30, 45, 60, 75, 90.

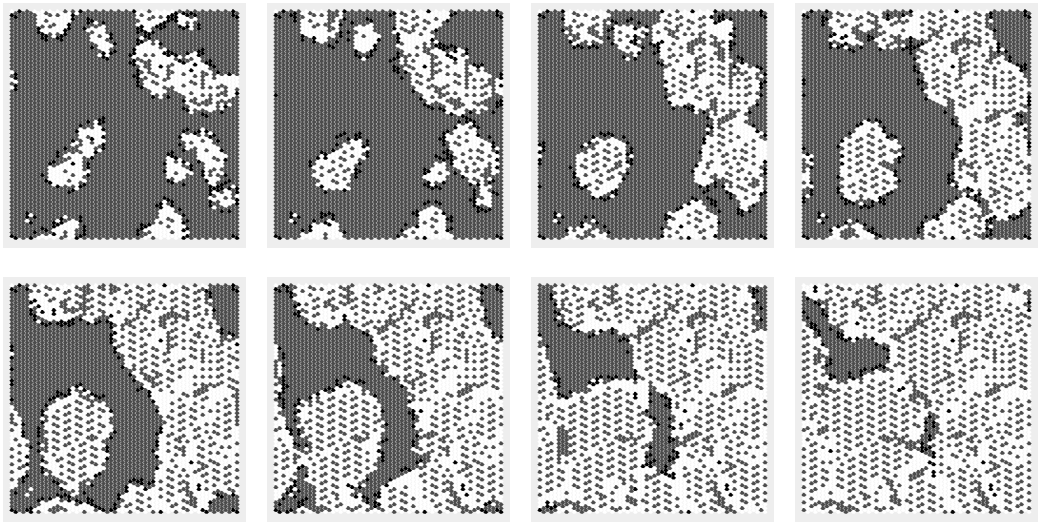

Figure 4. Snapshots of the configurations produced by rule 121221221221112000002111120 at iterations 15, 30, 45, 60, 75, 90

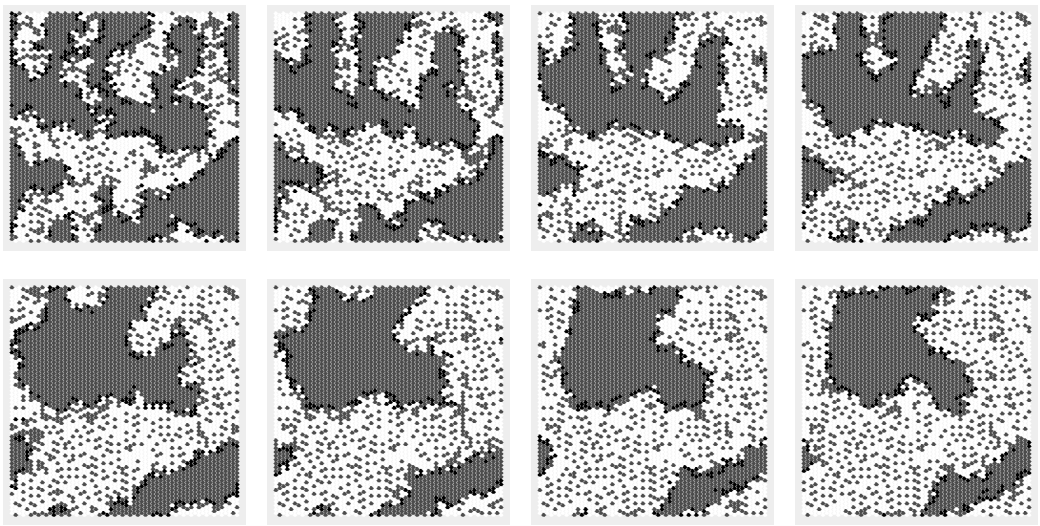

Figure 5. Snapshots of the configurations produced by rule 121222211220212100100111102 2 at iterations 15, 30, 45, 60, 75, 90.

Looking at this snapshot, it is straightforward that the configurations generated by the rules found by the proposed framework could be considered "similar" to the ones generated by the "burning paper" rule by a human being. In particular, all the rules have generated configurations that respect the two properties required: white cells tend to "stick together" (i.e. they usually form more or less compact groups) and black cells are localized mostly on the edge of the white aggregations. In all the GA executions that we have performed, the maximum number of iterations does not seem to influence the performances of the

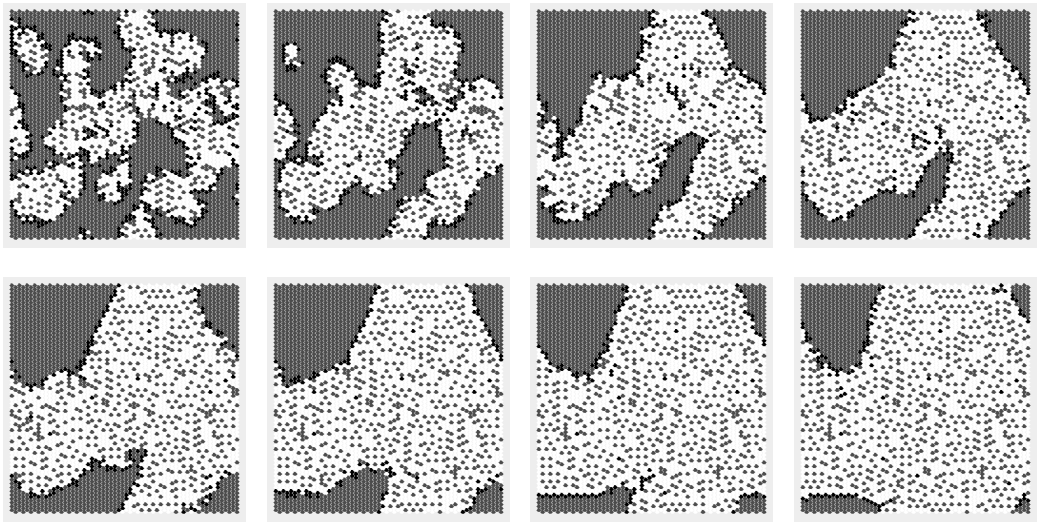

Figure 6. Snapshots of the configurations produced by rule 12122222200002110010111111111 at iterations 15, 30, 45, 60, 75, 90.

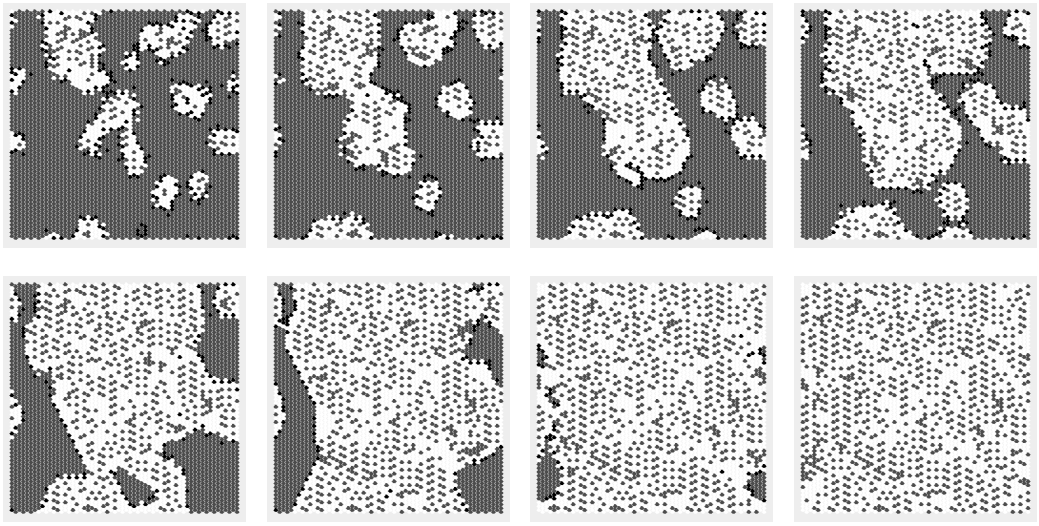

Figure 7. Snapshots of the configurations produced by rule 12222122212000201100111111120 at iterations 15, 30, 45, 60, 75, 90.

search algorithm as much as the number of individuals in the GA population: in almost all the cases, the GA found the definitive solution within the first 25 or 30 iterations.

Furthermore, it is interesting to note that, surprisingly, most of the experiments (roughly more than 85% of the GA runs) resulted in rules that produced either a rapidly expanding population of white cells or a population of black ones that rapidly consumes the whites[5]. This is particularly interesting since in

[5]The rest of the experiments usually led to rules that tend to quicly establish oscillating patterns.

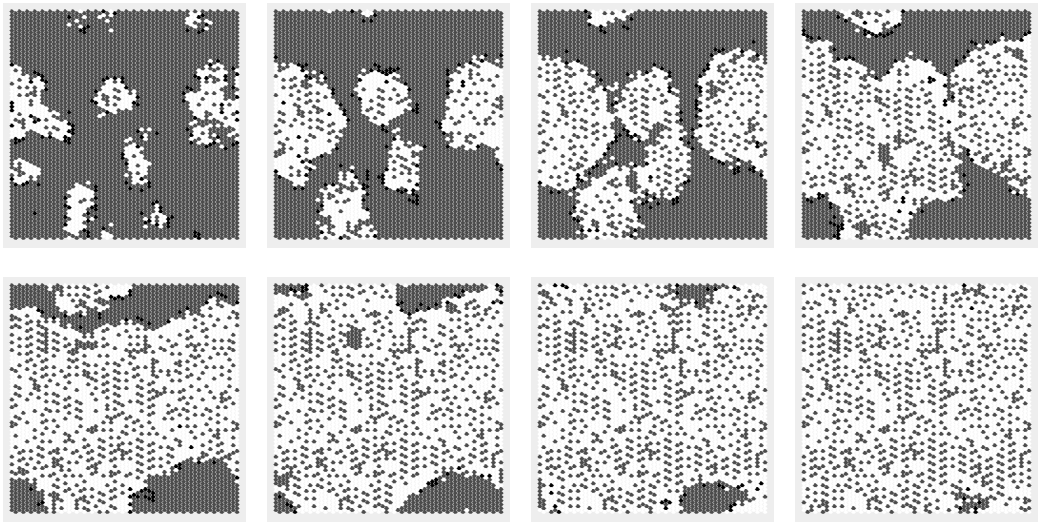

Figure 8. Snapshots of the configurations produced by rule 12222221022110001110111120 11 at iterations 15, 30, 45, 60, 75, 90.

this work we focused on *static* patterns, and we did not expect to necessarily find this kind of dynamic behaviour by examining only isolated snapshots. This could reveal that there is some deep correlation between the patterns generated by a CA and the kind of dynamics it produces.

7. Conclusions and Future Work

A generic and modular framework based on Genetic Algorithms (GAs) to search for rules that generate some specific patterns has been presented in this work. The approach is said to be generic because the nature of the target family of rules can be indirectly specified using a particular component, called *evaluator*, whose task is to assign to each configuration a real number that measures how "good" a rule is with respect to the class we are looking for. In this paper, as a proof of concept, the rule class to look for was defined indirectly, starting from another rule: we were interested in finding rules that are able to generate patterns such as the ones generated by a specific rule called the "burning paper" rule. We assumed the rule to be unknown, and defined an evaluator to recognise (i.e. assign an high score to) the patterns that resembled the ones generated by this specific one.

We considered the possibility to use as evaluators many different kinds of classifiers, but experimental results showed that two of them are the most promising: one based on Support Vector Machines (SVM), and another one based on a Artificial Neural Networks (ANN) and, in particular, Multilayer Perceptrons. To develop an efficace evaluator module, we had to choose: wether to use SVMs or ANNs, an appropriate training set and a satisfactory format for the instances to be classified (i.e. the choice of the attributes to provide the classifier with in order to allow it to perform classification). In order to choose the most appropriate values for these three, closely related, aspects, we tested various combinations of classifiers, instance formats and training sets. The ANNs based method, with particular features and input data, returned outstandingly good results, with a classification accuracy of above 99%. The

resulting model was then used as the evaluator component, and included in the calculation of the fitness function of the GA. The generic nature framework, however, allows the user to change the module being used as evaluator with minimal effort. This could allow, for example, to try several other classifiers and choosing the best one for the problem being faced, or even to try using hand written algorithms to assign a score to the configurations. The results of the search performed by the overall system were particularly encouraging. Not only we were able to find rules that generate configurations that exhibit the desired features, but, in the wide majority of cases, the dynamical behaviour of the rules that resulted from the GA-based search resembled the ones of the "burning paper" rule (e.g. a rapidly expanding population of white cells or a population of black ones that rapidly consumes the whites).

The results described in this paper are, in our opinion, promising: not only we were able to develop a Machine Learning model that is able to recognise the family of patterns we were considering, but we also managed to find similar rules using the GA search. This should pave the way to further research using the same methodology and possibly generalising it and applying it to real-life pattern recognition applications.

Acknowledgments

We gratefully acknowledge Sara Manzoni for her precious hints and support and the Laboratory of Artificial Intelligence (LINTAR) of the University of Milano-Bicocca for allowing us the development of the experiments presented in this paper.

References

[1] Andre, D., Bennett III, F. H., Koza, J. R.: Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem, *Genetic Programming 1996: Proceedings of the First Annual Conference* (J. R. Koza, D. E. Goldberg, D. B. Fogel, R. L. Riolo, Eds.), The MIT Press, Cambridge, MA, 1996.

[2] Bandini, S., Manzoni, S., Redaelli, S., Vanneschi, L.: Automatic Detection of Go-Based Patterns in CA Model of Vegetable Populations: Experiments on Geta Pattern Recognition, *Proceedings of the Seventh International Conference on Cellular Automata for Research and Industry (ACRI 2006)* (S. E. Yacoubi, B. Chopard, S. Bandini, Eds.), Lecture Notes in Computer Science, LNCS 4173, Springer, Berlin, Heidelberg, New York, 2006, ISBN 3-540-40929-7.

[3] Bandini, S., Manzoni, S., Redaelli, S., Vanneschi, L.: Emergent Spatial Patterns in Vegetable Population Dynamics: towards Pattern Detection and Interpretation, *Modelling Complex Systems by Cellular Automata (MCSCA 2006), workshop of the* 6^{th} *International Conference on Computational Science (ICCS 2006)* (V. N. A. *et al.*, Ed.), 3, Springer, Berlin, Heidelberg, New York, 2006.

[4] Bilotta, E., Lafusa, A., Pantano, P.: Is self-replication an embedded characteristic of artificial/living matter?, *Artificial Life VIII*, 2002, 38–48.

[5] Bilotta, E., Lafusa, A., Pantano, P.: Searching for complex CA rules with GAs, *Complexity*, **8**(3), 2003, 56–67.

[6] Boser, B., Guyon, I., Vapnik, V.: A training algorithm for optimal margin classifiers, *Proceedings of the fifth annual workshop on Computational learning theory*, 1992, 144–152.

[7] Brady, M. L., Raghavan, R., Slawny, J.: Probabilistic Cellular Automata in Pattern Recognition, *Proc. Inter. Joint Conf. on Neural Networks*, 1989.

[8] Breukelaar, R., Bäck, T.: Using a genetic algorithm to evolve behavior in multi dimensional cellular automata: emergence of behavior, *Genetic and Evolutionary Computation Conference, GECCO 2005, Proceedings* (H. G. Beyer, U. M. O'Reilly, Eds.), ACM, Washington DC, USA, 2005.

[9] Chang, C.-C., Lin, C.-J.: *LIBSVM: a library for support vector machines*, 2001, Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[10] Chopard, B., Droz, M.: *Cellular Automata Modeling of Physical Systems*, Cambridge University Press, Cambridge, UK, 1998.

[11] Cortes, C., Vapnik, V.: Support-vector networks, *Machine Learning*, **20**(3), 1995, 273–297.

[12] Das, R., Crutchfield, J. P., Mitchell, M.: Evolving globally synchronized cellular automata,, *Proceedings of the Sixth International Conference on Genetic Algorithms* (L. J. Eshelman, Ed.), Morgan Kaufmann, San Francisco, CA, 1995.

[13] El-Manzalawy, Y., Honavar, V.: *WLSVM: Integrating LibSVM into Weka Environment*, 2005, Software available at `http://www.cs.iastate.edu/~yasser/wlsvm`.

[14] Ganguly, N., Maji, P., Dhar, S., Sikdar, B., Chaudhuri, P.: Evolving Cellular Automata as Pattern Classifier, *Proceedings of Fifth International Conference on Cellular Automata for Research and Industry, ACRI 2002, Switzerland*, 2002, 56–68.

[15] Goldberg, D.: *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[16] H. Juillé, J. B. P.: Coevolutionary learning: a case study, *ICML '98 Proceedings of the Fifteenth International Conference on Machine Learning*, Morgan Kaufmann, San Francisco, CA, 1998.

[17] Holland, J.: *Adaptation in natural and artificial systems*, University of Michigan Press, 1975.

[18] Hordijk, W.: Correlation analysis of the synchronizing-ca landscape, *Physica D*, **107**, 1997, 225–264.

[19] Hsu, C.-W., Chang, C.-C., Lin, C.-J.: A Practical Guide to Support Vector Classification, available at `http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`, 2003.

[20] Ibarra, O., Palis, M., Kim, S.: Fast parallel language recognition by cellular automata, *Theoretical Computer Science*, **41**(2-3), 1985, 231–246.

[21] J. P. Crutchfield, M. Mitchell, R. D.: Evolutionary design of collective computation in cellular automata, *Evolutionary Dynamics: Exploring the Interplay of Selection, Accident, Neutrality, and Function* (J. P. Crutchfield, P. Schuster, Eds.), Oxford University Press, Oxford, UK, 2003.

[22] Koza, J.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, USA, 1992, ISBN 0-262-11170-5.

[23] Land, M., Belew, R. K.: No perfect two-state cellular automata for density classification exists, *Physical Review Letters*, **74**(25), 1995, 5148–5150.

[24] Maji, P., Shaw, C., Ganguly, N., Sikdar, B. K., Chaudhuri, P. P.: Theory and application of cellular automata for pattern classification, *Fundam. Inf.*, **58**(3-4), 2003, 321–354, ISSN 0169-2968.

[25] Makatchev, M., Lang, S. Y. T.: On the Complexity of Image Processing and Pattern Recognition Algorithms, *Proc. of the Int. Workshop on Image, Speech, Signal Processing and Robotics*, 1998.

[26] Mitchell, M., Crutchfield, J., Das, R.: Evolving cellular automata with genetic algorithms: A review of recent work, *Proceedings of the First International Conference on Evolutionary Computation and Its Applications (EvCA96)*, 1996.

[27] Mitchell, M., Crutchfield, J. P., Hraber, P. T.: Evolving cellular automata to perform computations: Mechanisms and impediments, *Physica D*, **75**, 1994, 361–391.

[28] Mitchell, M., Hraber, P. T., Crutchfield, J. P.: Revisiting the edge of chaos: Evolving cellular automata to perform computations, *Complex Systems*, **7**, 1993, 89–130.

[29] Orovas, C.: *Cellular Associative Neural Networks for Pattern Recognition*, Ph.D. thesis, University of York, UK, 1999, Downlodable version at: http://citeseer.ist.psu.edu/orovas99cellular.html.

[30] Raghavan, R.: Cellular automata in pattern recognition, *Information Sciences*, **70**(1), 1993, 145–177.

[31] Rumelhart, D., Hinton, G., Williams, R.: *Learning internal representations by error propagation*, MIT Press Cambridge, MA, USA, 1986.

[32] Sipper, M.: Non-Uniform Cellular Automata: Evolution in Rule Space and Formation of Complex Structures, *Proceedings of the 4th International Workshop on the Synthesis and Simulation of Living Systems (Artificial Life IV)* (Brooks, R. A, Maes, Pattie, Eds.), MIT Press, 1994.

[33] Sipper, M.: Co-evolving non-uniform cellular automata to perform computations, *Physica D*, **92**(3-4), 1996, 193–208.

[34] Sipper, M.: *Evolution of parallel cellular machines: The Cellular Programming Approach*, Springer New York, 1997.

[35] Sipper, M.: The evolution of parallel cellular machines: Toward evolware, *BioSystems*, **42**, 1997, 29–43.

[36] Sipper, M., Tomassini, M., Capcarrere, M.: Evolving asynchronous and scalable non-uniform cellular automata, *Proceedings of International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA97)*, 1997, 66–70.

[37] Smith III, A.: Real-time language recognition by one-dimensional cellular automata, *Journal of Computer and System Sciences*, **6**(3), 1972, 233–253.

[38] Tzionas, G. P., Tsalides, P. G., Thanailakis, A.: Cellular-automata-based learning network for pattern recognition, *Proc. of SPIE Vol. 1606*, 1991.

[39] Vapnik, V.: *Estimation of Dependences Based on Empirical Data [in Russian]*, Nauka, Moscow, 1979, English translation: Springer Verlag, New York, 1982.

[40] Verel, S., Collard, P., Tomassini, M., Vanneschi, L.: Fitness Landscape of the Cellular Automata Majority Problem: View from the "Olympus", *Theoretical Computer Science*, **378**(1), 2007, 54–77.

[41] Witten, I., Frank, E.: *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd edition, Morgan Kaufmann, San Francisco, 2005.

[42] Wolfram, S.: Statistical mechanics of cellular automata, *Reviews of Modern Physics*, **55**(3), 1983, 601–644.

[43] Wolfram, S.: *A New Kind of Science*, Wolfram Media, 2002.

[44] Wuensche, A.: Classifying Cellular Automata Automatically, *Complexity*, **4**(3), 1999, 47–66.

[45] Wuensche, A.: Self-reproduction by glider collisions: the beehive rule, *International Journal Pollack et. al*, 2004, 286–291.

[46] Wuensche, A.: Discrete Dynamics Lab (DDLab), www.ddlab.com and http://www.cogs.susx.ac.uk/users/andywu/ddlab.html, November 2005.

[47] Wuensche, A., Adamatzky, A.: On Spiral Glider-Guns in Hexagonal Cellular Automata:. Activator-Inhibitor Paradigm, *International Journal of Modern Physics C*, **17**(07), 2006, 1009–1026.