Cellular Automata Pattern Recognition and Rule Evolution through a Neuro-Genetic approach

Stefania Bandini[(1)], Leonardo Vanneschi[(1)], Andrew Wuensche[(2)], and Alessandro Bahgat Shehata[(1)*]

(1) *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*
(2) *Dept. of Informatics, School of Science and Technology, University of Sussex, UK*
{{*bandini, vanneschi, bahgat*}*@csai.disco.unimib.it, andy@ddlab.org*}

Abstract. Cellular Automata rules often produce spatial patterns which make them recognizable by human observers. Nevertheless, it is generally difficult, if not impossible, to identify the characteristic(s) that make a rule produce a particular pattern. Discovering rules that produce spatial patterns that a human being would find "similar" to another given pattern is a very important task, given its numerous possible applications in many complex systems models. In this paper, we propose a general framework to accomplish this task, based on a combination of Machine Learning strategies including Genetic Algorithms and Artificial Neural Networks. This framework is tested on a 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 our framework in real-life complex systems models.

1 Introduction

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 [17]. 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 [9, 1, 2, 4, 14]). In all those studies, however, the objective was to find CA rules that performed precise tasks. None of these studies focused on finding rules able to generate

* *Corresponding author: Leonardo Vanneschi, Department of Informatics, Systems and Communication (D.I.S.Co.), University of Milano-Bicocca, 20126 Milan. Tel: +390264487874. Fax: +390264487805. Email: vanneschi@disco.unimib.it.*

given spatial *patterns*, which is the objective here. In particular, in this paper we present a framework that allows searching for rules that generate (imitate), in their dynamics, certain families of patterns.

The use of CA as an instrument to perform pattern recognition is not new (see for instance [13, 8, 11, 10, 5]). However, these contributions are very far from the scope of this paper, where we try to address the problem from the opposite viewpoint: we do not use CA to recognize patters, but we look for an instrument to recognise the patterns generated by CA. In other words, the rules found by our framework should be able to produce configurations that a human observer could consider, in some generic sense, *similar* to the ones generated by a given target rule.

Among other factors, the huge size of the space of all the possible rules and the fact that the behaviour of a CA rule is not easy to predict just by looking at its syntactical representation (two rules with extremely similar representations can result both in almost identical or incredibly different global dynamics [19]), led us to choose Genetic Algorithms (GAs) [7, 6] 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 [19]. 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. 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. In particular, we experimentally found Artificial Neural Networks trained with a Backpropagation learning algorithm [12] (ANNs from now on) 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 our 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 a prototype a particular rule called the "burning paper" rule, discovered by Wuensche [20]. Figure 1 shows an example snapshot. Details about the rule can be found at `http://www.cogs.susx.ac.uk/users/andywu/multi_value/dd_life.html` under "predator-prey dynamics". In this rule, cells can be in three possible states usually represented by white, red and black colors and the dynamics of this rule

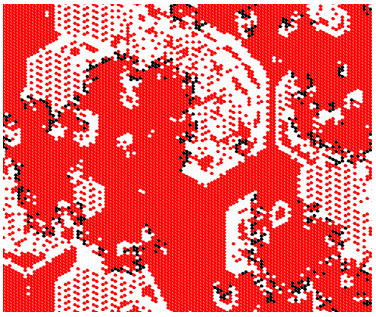

Fig. 1. Snapshot of a configuration generated by the rule we are considering as prototype. (`k-code 1202022101201121112121110101`).

can be informally described by the following two properties: white cells tend to form more or less compact, growing, aggregations, while black cells are localized mainly on the edges and grow wave-like into the white aggregations. Red cells are considered as a background. This rule will not be further discussed here to save space, but the reader is referred to [20] for an introduction.

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

2 Structure and Functioning of our 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 [7, 6], including the initialization of a population of (typically randomly 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 the population and the application of genetic operators such as selection, crossover and mutation. Potential solutions contained into the population are CA rules coded as strings of characters in a way that closely resembles the one used in [16]. In particular, each rule can be identified by a string, called *k-code*. In our case, therefore, rules are represented as 28 character strings, whose alphabet is determined by the possible state values (i.e. `0`, `1` or `2`, representing white, red and black cells of the "burning paper" rule).

The main difference between our framework and a standard GA is in the fitness evaluation. In order to define the fitness of each GA individual, we need

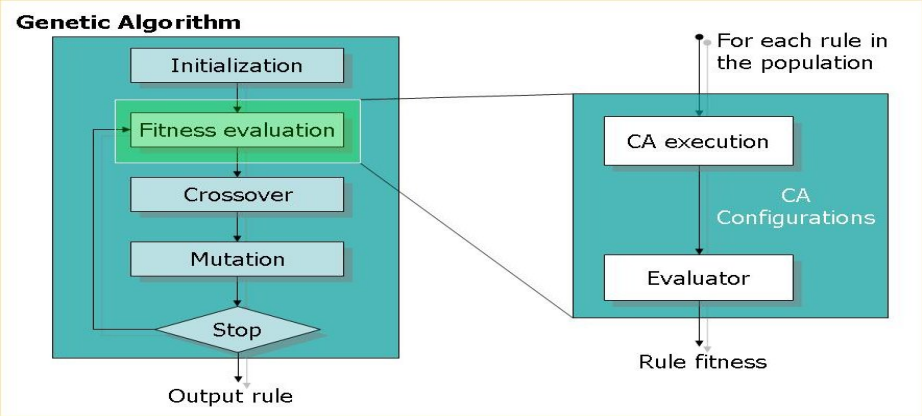

Fig. 2. A graphic representation of the general architecture of the framework presented in this paper.

an instrument able to distinguish the patterns that show some *similarity* with the prototype patterns from those that do not. In particular, we need a function able to provide us with a numeric value, that we could interpret as the probability that a configuration belongs to a given class. In the following Section, we will describe our analysis of ANNs for this task. For now, however, we simply assume that we have generated a *model* 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 starting configurations, in order to reduce the risk of misclassification due to spurious dynamics caused by some particular choices of the initial state. 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
             lattice.step(S); fitness := fitness + model(lattice);
       endfor
endfor
fitness := fitness/(R*C);
return(fitness);
```

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 according to 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.

3 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 [15] implementation for our Machine Learning methods, since it includes many different classifiers. We performed many experiments but we describe only the ones that have returned the most promising results: models based on ANNs. In order to quantify to what extent the ANN results improved on other methods, we also present a comparison of ANNs with Support Vector Machines (SVMs) [3].

When ANNs are trained, the cardinality of the input configurations 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 networks trained with the Backpropagation learning rule. For this reason, instead of having an attribute for each lattice cell, we tried to encode the *spatial* information about the configurations using the concept of neighbourhood. Wuensche in [18] used the neighbourhood frequencies (*rule-table look-up* frequencies, or *k-block* frequencies) to calculate the input-entropy and then classify the behaviour of CA. Therefore, we tried to use the neighbourhood counts as a set of features for classification, hoping that it would result in better classification and in particular that it would allow the ANN to distinguish configurations that have the same state densities as the targets, but with different patterns. The new instances, therefore, are 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.

As parameters used to generate the ANN models, we used the default values provided by the Weka software environment (see [15] for a discussion and motivation). The model was generated from a mixed training set composed of 8000 instances: 50% of them was composed by hand-made configurations that

"resemble" the ones typically generated by the "burning paper" rule (tagged with **Y**), 25% from totally different, randomly generated configurations (tagged with **N**) and the remaining 25% from a set of configurations tagged with **S** of configurations that do not resemble configurations found by the "burning paper" rule, but have the same global state densities of the configurations in class **Y**. The instances generated from configurations in **S** were added to the training set in order to make it harder, for the classifier, to heavily rely on state densities to discriminate between classes.

The experimental results, obtained using 10-fold cross-validation, were impressive: the ANN-based model was able to correctly classify 99.26% of the instances against, for instance, 50.73% obtained by Support Vector Machines (SVMs). Many other classifiers contained in Weka have been tested, but all of them performed approximately as the SVMs and much worse than ANNs (these results are not shown here to save space). Table 3 reports a comparison of the performances of ANNs and SVMs (where also for SVMs we used the default parameter setting proposed by Weka). These results are encouraging and pave the way for the use of the models generated by ANNs to calculate the fitness of GAs individuals.

	SVMs	**ANNs**
accuracy	50.73%	99.26%
precision (**Y**)	0.504	0.99
recall (**Y**)	1	0.996
precision (**N**)	1	0.995
recall (**N**)	0.015	0.99

Table 1. Comparison between the results of the models based on Support Vector Machines (SVMs) and Artificial Neural Networks (ANNs), calculated using 10-fold cross validation

4 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. The structure of the GA we used to evolve CA rules is fairly standard [7, 6]: we used Tournament Selection of size 20, a crossover rate of 0.95, a mutation rate of 0.001, a maximum number of generation equal to 100 and a fixed population size of 600 individuals. Furthermore, the initial populations we considered were composed of 600 randomly generated rules.

We have performed 500 indipendent GA executions. Table 2 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

	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 2. 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.

strings that represent some of the rules obtained from experimentation it emerges that some of the rule components (i.e. neighbourhoods) are more important than others in determining the nature of the behaviour of a rule. This is not a complete surprise: Wuensche and Adamatzky [21] 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 2, it is possible to extract a sort of *schema* [7, 6], 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 [7, 6] for a detailed discussion of schema theory and its importance in GAs).

In order to be able to compare the configurations generated by our framework with the ones generated by the "burning paper" rule, we have integrated our framework with DDLab [20]. Figures 3 and 4 report the snapshots (obtained with DDLab) of some of the configurations generated by two of the rules reported in table 2 (the ones marked with a "∗" in the table). We do not report snapshots 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" to the ones shown in these figures. Looking at these snapshots and comparing them with the one reported in figure 1, we can notice that they are "similar" to the ones generated by the "burning paper" rule. In particular, all the rules have generated configurations that respect the following two properties: white cells

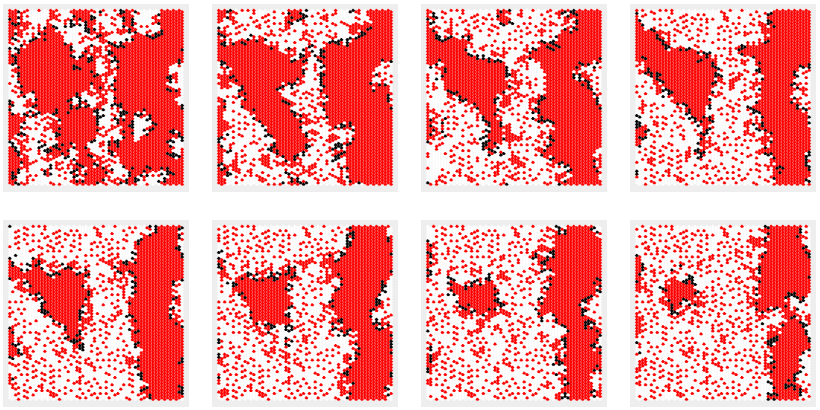

Fig. 3. Snapshots of the configurations produced by rule 12121222102021110101111111111 at iterations 15, 30, 45, 60, 75, 90.

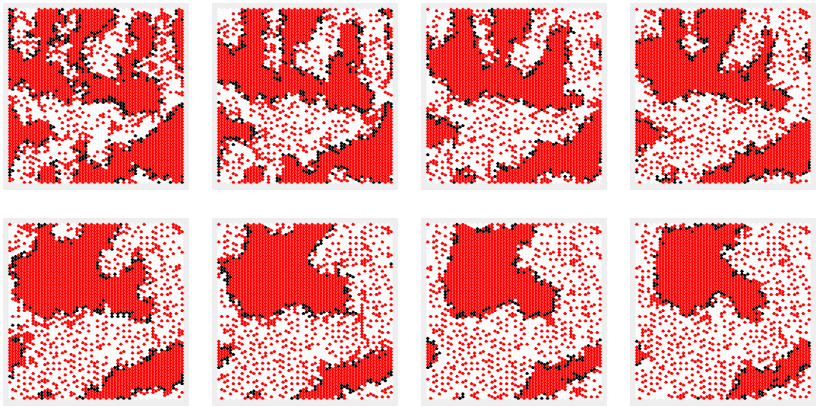

Fig. 4. Snapshots of the configurations produced by rule 12122221122021210010011111022 at iterations 15, 30, 45, 60, 75, 90.

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.

5 Conclusions and Future Work

A generic and modular framework 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 considered the possibility to use as evaluators many different kinds of classifiers, but experimental results showed that Artificial Neural Networks (ANNs) trained with the Backpropagation learning rule are the most promising ones.

To develop an efficient evaluator module, we had to choose 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 two aspects, we tested various combinations of 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 a Genetic Algorithm (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 were we 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). 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. The snapshots in figures 1, 3 and 4 were created with DDLab [20].

References

1. D. Andre, F. H. Bennett III, J. R. Koza, Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem, in: J. R. Koza, D. E. Goldberg, D. B. Fogel, R. L. Riolo (Eds.), Genetic Programming 1996: Proceedings of the First Annual Conference, The MIT Press, Cambridge, MA, 1996, pp. 3–11.
2. R. Breukelaar, T. Bäck, Using a genetic algorithm to evolve behavior in multi dimensional cellular automata: emergence of behavior, in: H.-G. Beyer, U.-M. O'Reilly (Eds.), Genetic and Evolutionary Computation Conference, GECCO 2005, Proceedings, Washington DC, USA,, ACM, 2005, pp. 107–114.

3. C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
4. J. P. Crutchfield, M. Mitchell, R. Das, Evolutionary design of collective computation in cellular automata, in: J. P. Crutchfield, P. Schuster (Eds.), Evolutionary Dynamics: Exploring the Interplay of Selection, Accident, Neutrality, and Function, Oxford University Press, Oxford, UK, 2003, pp. 361–411.
5. N. Ganguly, P. Maji, S. Dhar, B. Sikdar, and P. Chaudhuri. Evolving Cellular Automata as Pattern Classifier. *Proceedings of Fifth International Conference on Cellular Automata for Research and Industry, ACRI 2002, Switzerland*, pages 56–68, 2002.
6. D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.
7. J. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, 1975.
8. O. Ibarra, M. Palis, and S. Kim. Fast parallel language recognition by cellular automata. *Theoretical Computer Science*, 41(2-3):231–246, 1985.
9. M. Mitchell, J. P. Crutchfield, P. T. Hraber, Evolving cellular automata to perform computations: Mechanisms and impediments, Physica D 75 (1994) 361–391.
10. C. Orovas. *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.
11. R. Raghavan. Cellular automata in pattern recognition. *Information Sciences*, 70(1):145–177, 1993.
12. D. Rumelhart, G. Hinton, and R. Williams. *Learning internal representations by error propagation*. MIT Press Cambridge, MA, USA, 1986.
13. A. Smith III. Real-time language recognition by one-dimensional cellular automata. *Journal of Computer and System Sciences*, 6(3):233–253, 1972.
14. S. Verel, P. Collard, M. Tomassini, and L. Vanneschi. Fitness landscape of the cellular automata majority problem: View from the "Olympus". *Theoretical Computer Science* 378 (1), 2006, pp. 54–77.
15. I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005.
16. S. Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55(3):601–644, 1983.
17. S. Wolfram, A New Kind of Science, Wolfram Media, 2002.
18. A. Wuensche. Classifying Cellular Automata Automatically. *Complexity*, 4(3):47–66, 1999.
19. A. Wuensche. Glider dynamics in 3-value hexagonal cellular automata: the beehive rule. *International Journal of Unconventional Computing*, 1(4):375–398, 2005.
20. A. Wuensche. Discrete dynamics lab (ddlab). www.ddlab.com and http://www.cogs.susx.ac.uk/users/andywu/ddlab.html, November 2005.
21. A. Wuensche and A. Adamatzky. On Spiral Glider-Guns in Hexagonal Cellular Automata:. Activator-Inhibitor Paradigm. *International Journal of Modern Physics C*, 17(07):1009–1026, 2006.