

# Evolution of Cellular-automaton-based Associative Memories

Marcin Chady and Riccardo Poli  
School of Computer Science  
The University of Birmingham (UK)  
email: {mcc, rmp}@cs.bham.ac.uk  
Technical Report no. CSRP-97-15  
May 1997

## Abstract

Cellular Automata (CAs) are discrete dynamic systems composed of a large set of simple units organised into a regular one-, two- or multi-dimensional grid which update their state on the basis of their previous state and the state of a small number of neighbouring cells. CAs have been traditionally used for image processing and for hydrodynamics, thermodynamics and turbulence modelling. More recently CAs have also been used as mechanisms to study emergent computation, the phenomenon in which a large set of simple interacting elements with little information produce a complex coordinated information processing behaviour. In this paper, using the power of genetic algorithms, we study the ability of CAs to perform two very important forms of emergent computation: pattern association and associative memory.

## 1. Introduction

A cellular automaton is a discrete system which evolves in discrete space and time [1, 2]. It consists of a large number of cells which are organised in the form of an  $n$ -dimensional lattice. Each cell can be seen as a single processor which communicates only with the neighbouring cells and updates its state according to information received from them, as well as to its own present state. The state-update function is the same for all cells in the lattice. Although this is usually a very simple rule, on a global scale it can produce a very complex and often unpredictable behaviour.

Thanks to this complex behaviour CAs are an attractive platform for studying and implementing efficient forms of emergent computation, the phenomenon in which a large set of simple interacting elements with little information produce a complex, coordinated information processing behaviour. Interesting results with one-dimensional binary CAs have been obtained, for example, by Packard [3], Mitchell *et al.* [4, 5] and Andre *et al.* [6], who used evolutionary algorithms to discover CA rules which solve large-scale majority-classification and synchronisation problems.

In the work presented in this paper we also use binary CAs and evolutionary algorithms to discover state-update rules, but we concentrate on a considerably hard problem: the emergence of pattern-association and auto-association behaviour. In addition, differently from the previous work, we look at a particular class of CAs, which we term *feed-forward* CAs. Feed Forward CAs (FFCAs) are CAs with an update rule which yields a directional flow of information by using a neighbourhood which extends only in one direction, like in Figure 1.1.

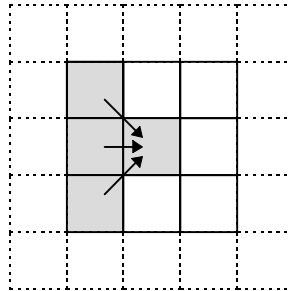


Figure 1.1 An example neighbourhood for a feed forward CA.

Two-dimensional FFCAs can be easily transformed into devices which perform global computation if we consider the left-most column as the input interface and the right-most column as the output interface, like in Figure 1.2.

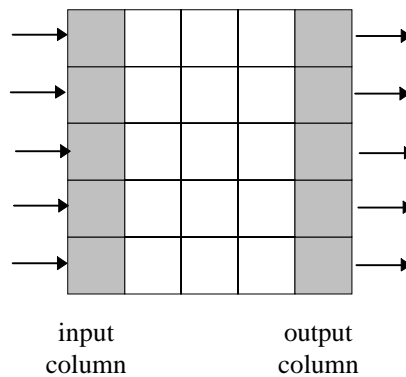


Figure 1.2 An example architecture for a CA-based pattern associator.

In our work we consider the upper and lower edges of the lattice as joined together to form a cylinder, so that information can pass freely between the top and bottom rows. The input and output columns, on the contrary, are not connected, so that information placed in the input column remains unchanged in the subsequent time steps, while information produced on the output column is discarded from one step to the next.

With this kind of CAs it is possible to map binary input vectors into corresponding output vectors, and, therefore, to implement pattern associators. In this paper we will mainly concentrate on a special class of pattern associators, namely the class of auto-associators or associative memories, where the CA is expected to retrieve incomplete binary vectors or rectify distorted ones, in a fashion similar to the operation of a Hopfield neural network [7].

In the next section we describe our approach to discovering CAs with emergent associative-memory properties, and in Section 3 we report the results obtained. Section 4 discusses these results, and Section 5 draws some conclusions and outlines some ideas for future research.

## 2. Evolving FFCAs

In order to make a CA perform a predefined global computation, i.e. to “program” it, one has to specify its initial configuration and the local rule that yields the required global behaviour. However, so far no method has been proposed to explain how to map the space of possible CA behaviours onto the corresponding local rules. In the literature on emergent computation with one-dimensional CAs, the rule table of binary CAs is usually encoded as a binary string and a genetic algorithm is used to evolve tables with the desired properties.

In our work we used the same strategy, but in addition to the rule table we also encoded in the chromosome the initial state of the CA. This may be very important in CAs for pattern association, as by modifying the behaviour of each cell in the automaton we allow the exploitation of local structure in the input/output patterns. A sample automaton and the corresponding chromosome are shown in Figure 2.1. The initial configuration of the input cells is not encoded, as these are always overwritten when an input is supplied.

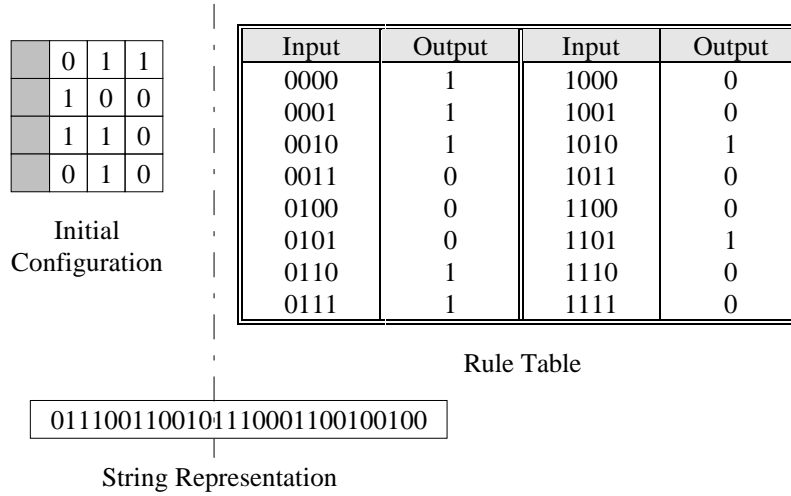


Figure 2.1 An example CA encoded in a binary string.

In initial experiments aimed at tuning the GA and finding a good fitness function, we trained CAs to behave like hetero-associators, i.e. the input vectors were distinct from the corresponding output vectors. Some noise was applied to the input vectors, so that the CAs were trained to recognise distorted inputs. The input vectors in the training set were different enough from one another so that the noise could not make them indistinguishable.

The GA used one-point crossover, with both offspring preserved. The mutation rate was exactly one bit per chromosome. Each individual in the population was evaluated by running the corresponding automaton on a training set of input vectors and comparing the results obtained with the expected output vectors. The fitness of an individual was proportional to the number and severity of the errors made in each test. In the course of these experiments we found that the GA worked best with tournament selection and with the following fitness function:

$$f(s) = 1 - \frac{1}{n \cdot d_{\max}^2} \sum_{i=1}^n |V_i - V_i^t| (d_{\max} - d_i),$$

where  $V_i$  is the output vector obtained when the  $i$ -th input vector of the training set is applied to the CA,  $|V_i - V_i^t|$  is the Hamming distance between the actual output vector and the desired output vector, i.e. the error produced by the CA,  $d_{\max}$  is the maximum possible error, and  $d_i$  is the severity of input noise in test  $i$  (measured as the Hamming distance between the input vector and its distorted version fed into the input column of the CA). With this function CAs were less penalised for errors if their input was severely distorted.

Because the CAs were trained with noisy inputs, we decided to explore the effect of the generation gap, i.e. the percentage of population replaced by offspring at each generation, on the GA effectiveness. Experimentally we observed that the best results are obtained with small generation gaps. Indeed, the best generation gap was 30%. This can be explained as follows. Nondeterministic inputs introduce some amount of randomness and inaccuracy in the fitness estimation, because every individual is tested with a different set of inputs. It is likely that an individual's fitness estimation will be slightly different in each cycle of the genetic algorithm. Therefore, if an individual is estimated positively many times, it is more likely that it is indeed a good individual. A big generation gap shortens an individual's life and reduces its chances of being retested over many generations, thus reducing the generation gap gives greater stability of evolution and consequently leads to better results.

### 3. CA-based associative memory

The preliminary experiments mentioned above were quite promising and showed that genetic algorithms are an effective tool for producing FFCAs with pattern association behaviour. As a first step towards a rigorous study of the properties of FFCAs we decided to concentrate on their capability to act as associative memories. In the following subsections we report on our work towards an estimation of FFCA capacity to remember patterns.

### 3.1 Simple neighbourhood CA

In a first series of experiments on FFCA associative memories we used the neighbourhood shown in Figure 1.1. The task of rectifying a pattern may involve extracting global information that a single local rule using 4 cells cannot extract. It was hoped that the emergent behaviour of the CA would provide some means of global communication across the lattice, so that the whole pattern could be analysed. A helpful feature of FFCA is that the input pattern placed in the input column stays constant throughout the computation, providing a steady reference for the rest of the automaton.

Two series of experiments were run, using

- A. an 8×8 CA,
- B. a 16×16 CA.

In each series the genetic algorithm was run repeatedly with an increasing number of pattern pairs to learn. In each pattern pair the input and output vectors were the same, but 20% noise was applied to the input vectors. The GAs were run with the parameters shown in Table 3.1.

Parameter	A	B	Input/Output
CA Width	8	16	0000000000000000
CA Height	8	16	1111111111111111
Minimum fitness	0	0	0101010101010101
Minimum generations	50	50	1010101010101010
Population size	400	400	0011001100110011
Chromosome length	72	256	1100110011001100
state encoding takes	56	240	1111000011110000
rule table takes	16	16	0000111100001111
Input noise level	20%	20%	0000000011111111
Number of CA steps per test	until relaxation	until relaxation	1111111100000000
Number of tests per pattern	100	100	
Number of patterns	2 - 8	2 - 10	

Table 3.1 GA/CA parameters for the CA-capacity experiments. For the 8×8 CA the input/output patterns were truncated after 8 bits.

Every individual was evaluated by feeding a distorted input vector into the input column and observing the CA's behaviour until it settled into a stable state. Individuals that did not settle within a given number of steps or fell into a periodic cycle were penalised by assuming the maximum error.

The GA ran for the minimum number of generations indicated in the table and then until no further progress was achieved, i.e. both the average and the best fitness in the population did not improve from one generation to another. For each experimental setting 3 independent runs of the GA were performed.

Table 3.2 shows typical rule tables obtained in the experiments.

Number of patterns	8×8 CA	16×16 CA
2	0001001100111111	0010001101111111
3	0001111100000111	0010111101001111
4	0000111100001111	0011111100000011
5	0000111100001111	0000111100001111
6	0000111100001111	0000110011001111
7	0000111100001111	0000111100001111
8	0000111101011111	0000111100001111
9	xxxxxxxxxxxxxxxxxx	0000111100001111
10	xxxxxxxxxxxxxxxxxx	0000111100001111

Table 3.2 Rule tables obtained in the simple-neighbourhood associative memory experiment.

It can be seen that, as the number of patterns grows, in both series the CA rule tables converge into one particular form, namely “0000111100001111,” which is a simple copy rule that takes the state of the cell on the left and copies it into the state of the current cell.

The emergence of the copy rule is what we should expect to happen if the number of patterns stored in the CA exceeds its capacity, because the copy rule is the rule with the minimum error correction ability. This can be understood by considering the following equation

$$C = p \cdot \delta \Rightarrow \lim_{p \rightarrow \infty} \delta = \lim_{p \rightarrow \infty} \frac{C}{p} = 0,$$

where  $C$  is the total memory capacity of the CA,  $p$  the number of stored patterns and  $\delta$  is the error correction capability. The formula simply shows that with a growing number of stored patterns the expected error correction capability approaches 0. Therefore, we can use the performance of CAs running a copy rule as the reference for measuring the CA error correction capability, i.e. we can assume that any CA having a higher fitness than a copy rule has a nonzero error correction capability. Furthermore, we can find the CA memory capacity by looking at the point where the evolved CA update rules converge to the copy rule.

Since  $C = p \cdot \delta$ , the maximum number of patterns stored in the CA will depend on the error correction requested from the CA, or in other words, on the level of noise applied to the input vectors. In these experiments, as well as the following ones, this level was set to 20%, which means the CAs were required to recover one fifth of the distorted input pattern.

The diagrams in Figure 3.1 show the performance of the simple-neighbourhood CA, compared with the performance of the copy rule.

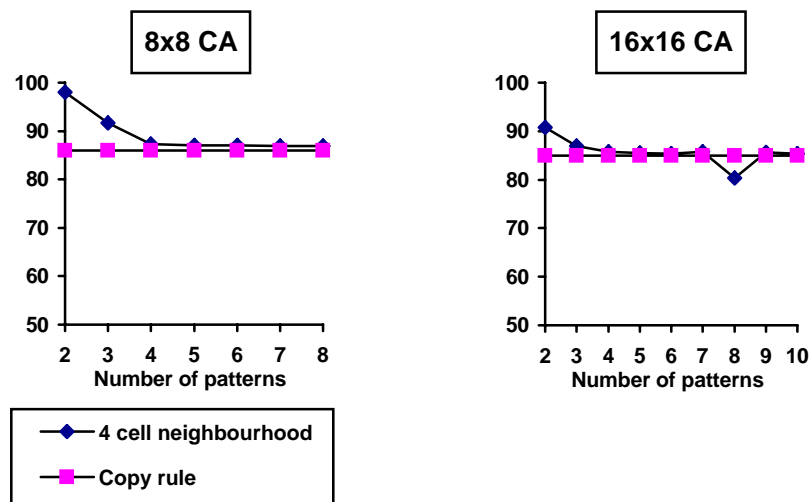


Figure 3.1 Performance of simple-neighbourhood CA compared with performance of the copy rule. The values denoted by diamonds correspond to averages of 3 runs of the GA.

From these results we can see that with up to 3 patterns the CA performance is above the copy rule. When the CA is required to store 4 patterns or more, the rules converge to the copy rule. This seems to indicate that the memory capacity is 3.

An interesting fact is that the 16x16 CA performance seems generally worse than 8x8 CA. A possible explanation for this effect is that the update rule with a small neighbourhood (i.e. 3+1 cells) is too simple to deal with 16-bit long patterns. Three new bits of information are processed by each cell in a single step, which is 37% of the total pattern length for the 8x8 CA, but only 19% for the 16x16 CA. Therefore, it might be relatively easy for a GA to discover rules that do better than the copy rule in a 8x8 CA, as the cells already have fairly large-scale information. On the contrary, the cells in a 16x16 CA really only have local information and therefore the GA has to solve the much harder task of discovering how to communicate information across the CA.

### 3.2 Increasing the CA capacity

The experiments described in the previous section were repeated with a neighbourhood including 6 cells, as shown in Figure 3.2.

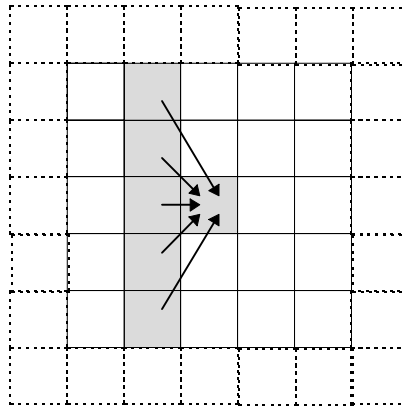


Figure 3.2 6-cell neighbourhood.

Since the rule tables for a 6-cell neighbourhood are quite large (they include  $2^6 = 64$  bits) and difficult to understand, they are not shown in the report. However, from Figure 3.3 it can be seen that the experimental results follow a similar pattern to the results in the previous section, i.e. up to a certain number of patterns (in this case 4) both kinds of CA exhibit a positive error correction capability. After that point, the update rules converge to the copy rule, or never reach a performance comparable with it.

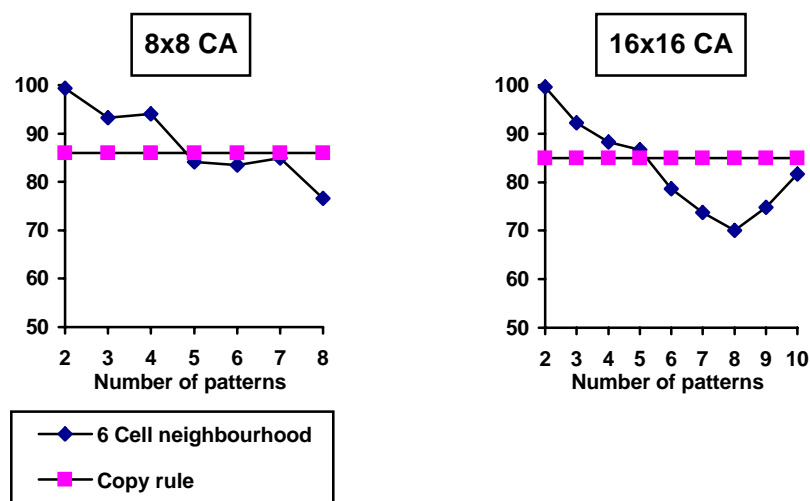


Figure 3.3 Performance of CAs with 6-cell neighbourhoods compared with the performance of the simple copy rule. The results are averages of 3 GA runs.

Comparing these results with those in the previous section one can see a clear increase in performance/capacity when using a bigger update rule neighbourhood. Both sizes of CA seem to have benefited from the bigger neighbourhood, although the 16x16 CA still does slightly worse than the 8x8 CA.

## 4. Discussion

It appears that increasing the size of CAs does not increase their memory capacity. In our experiments the capacity of the CA memory depends more on the update-rule neighbourhood size than on the size of the CA. Also, as it was pointed out in Section 3.1, the maximum number of patterns stored in the memory also depends on the level of noise applied to the input patterns. For this reasons, as well as because of the amount of time which would be required to carry out a more exhaustive investi-

gation (e.g. to try different set of patterns, bigger neighbourhoods, different noise levels, etc.), it is impossible to give here a precise estimation of CA memory capacity in relation to its architectural parameters, as it was done by Hopfield in [7] for neural network-based associative memories. However, the results of the above experiments suggest that CAs can indeed have properties of associative memory.

The reduced capability of the 16×16 CAs to store patterns when using 4-cell neighbourhoods might be due to the fact that the proportion of the input pattern analysed by a single cell (19% of the total length) is smaller than the amount of noise present in the input (20%). It could also be related to more generic properties of CAs that affect their pattern matching capabilities. Some analysis of the pattern matching properties of CA has been done by Jen [8] who showed that there is a minimal radius of the update rule neighbourhood for which one-dimensional CA can “recognise” an arbitrary input string and retain it as invariant for the next time step. Although the processing done by feed forward CAs is much more complex than the processing of one-dimensional CAs, and the number of patterns being recognised in our experiments is greater than one, Jen’s results can still be applied to the self-association problem in the FFCA, since at every column of a FFCA a certain amount of pattern recognition takes place.

## 5. Conclusions

In this paper we have introduced feed-forward cellular automata models and presented a way of “programming” them by means of a genetic algorithm to perform a computationally useful function, namely pattern association. The investigation presented here showed the applicability of genetic algorithms to two-dimensional FFCA programming and indicated the potential of these CAs to act as associative memories.

In order to produce a full evaluation of CA associative memories and their practical applications, more work is required. These are some of the important issues that need to be addressed:

- The interdependence between the number of patterns stored in the memory and its error correction capability. The experiments in section 3 will have to be repeated with different noise levels to see how they affect the CA memory performance.
- Our “until relaxation” terminating condition was the simplest way for deciding when the CA computations have finished. However, other kinds of CA responses might favour the emergence of well-performing individuals.
- In the previous sections CAs were trained to remember very specific patterns, disregarding the fact that their capacity might increase by carefully selecting the patterns being stored. Also, the process of adapting the CA to a particular set of patterns is very lengthy, as the whole evolution process has to be repeated for every different set of memories. An alternative approach would be to look for CAs that remember well many patterns and try to understand how they do that, so as to find a more analytical approach to CA programming. We could look at the dynamic behaviour of CAs and pick up the individuals that have a large number of broad basins of attraction. This would allow storing many patterns (attractors) and being able to rectify them from a wide range of distorted patterns (basins of attraction).

## 6. References

1. Toffoli, T., Margolus, N., *Cellular Automata Machines: A new Environment for Modelling*, MIT Press, London, 1987.
2. Wolfram, S., *Cellular Automata and Complexity: Collected Papers*, Addison-Wesley, 1994.
3. Packard, N.H., “Adaptation toward the edge of chaos,” in Kelso, J.A.S, Mandell, A.J., Shlesinger, M.F., *Dynamic Patterns in Complex Systems*, pages 293-301, World Scientific, Singapore, 1988.
4. Mitchell, M., Hraber, P.T, Crutchfield, J.P, “Revisiting the Edge of Chaos: Evolving Cellular Automata to Perform Computations,” *Complex Systems*, 7, 89-130, 1993.

5. Das, R., Mitchell, M., Crutchfield J.P., "A Genetic Algorithm Discovers Particle-Based Computation in Cellular Automata," *Proceedings of the Third Parallel Problem-Solving From Nature Conference*, March, 1994.
6. 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," in Koza, J.R, Goldberg, D.E., Fogel, D.B., Riolo, R.L., *Genetic Programming 1996: Proceedings of the First Annual Conference*, MIT Press, 1996.
7. Hopfield, J.J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proceedings of the National Academy of Sciences*, 79, 2554-2558, 1982.
8. Jen, E., "Invariant Strings and Pattern-Recognizing Properties of One-Dimensional Cellular Automata," *Journal of Statistical Physics*, 43, 1/2, 1986.