An Overview of Computation in Cellular Automata

Wim Hordijk

Karunya Institute of Technology and Sciences
Deemed University
Karunya Nagar, Coimbatore – 641 114
wim@santafe.edu

Introduction

Cellular Automata (CA) are mathematical models of decentralized spatially extended systems. They consist of a large number of relatively simple individual units, or "cells", which are connected only locally, without the existence of a central control in the system. Each cell is a simple finite automaton that repeatedly updates its own state, where the new cell state depends on the cell's current state and those of its immediate (local) neighbors. However, despite the limited functionality of each individual cell, and the interactions being restricted to local neighbors only, the system as a whole is capable of producing intricate patterns, and even of performing complicated computations. In that sense, they form an alternative model of computation, one in which information processing is done in a distributed and highly parallel manner. Because of these properties, CA have been used extensively to study complex systems in nature, such as fluid flow in physics or pattern formation in biology, but also to study information processing (computation) in decentralized spatially extended systems (natural or artificial). Here, we will give a brief overview of the different ways in which computations can be done with cellular automata.

The paper is organized as follows. First, the concept of cellular automata is introduced. Then some examples of specific computational tasks that can be performed with CA are presented. Next, different proofs of universal computation in CA are discussed. Then, an example of what is called "emergent computation" in CA is reviewed. Finally, some pointers to more information on cellular automata are provided.

Cellular Automata

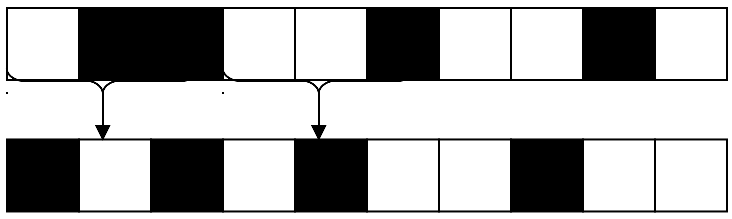
A *cellular automaton* (CA) consists of a regular grid (lattice) of cells (automata), each of which can be in one of a finite number of states. At discrete time steps, all cells simultaneously update their states depending on their current state and those of their immediate neighbors (i.e., depending on the local neighborhood configuration of each cell). For this update step, all cells use the same deterministic update rule, which lists the new cell states for each possible local neighborhood configuration. This update process is then repeated ("iterated") for a certain number of time steps.

In the simplest case, the CA lattice is a one-dimensional array of cells (1D CA), where each cell can be either black or white (two possible states), and the local neighborhood of a cell consists of the cell itself, the immediate neighbor to the left, and the immediate neighbor to the right (i.e., a *radius* of 1, or three cells total). So, there are $2^3 = 8$ possible neighborhood configurations for a cell, since each of the three cells in the local neighborhood can be either black or white. Such a 1D, two-state, radius 1 CA is known as an *elementary* CA (ECA). Of course many variations on this basic scheme exist, such as higher dimensional (2D, 3D, …) lattices, a larger number of states (>2), or larger neighborhood sizes (a radius of 2, 3, …). However, the basic principles (updating of cells based on local neighborhood configurations according to a given update rule) remain the same in all cases.

The example below shows one particular ECA update rule. The 8 possible local neighborhood configurations are listed in the first row (white is represented by a 0, and black is represented by a 1), and the new cell state for each neighborhood is given in the corresponding position in the second row.

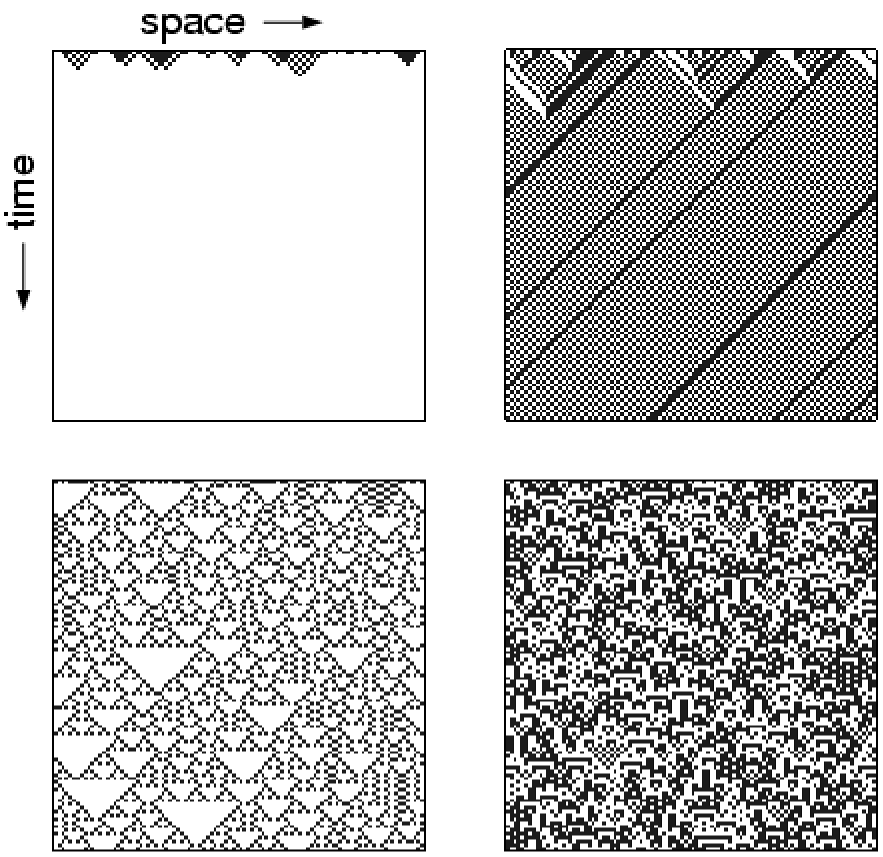
```
neighborhood: 111  110  101  100  011  010  001  000
new state:      0    1    1    0    0    0    1    0
```

For this particular update rule, the three neighborhoods “black, black, white” (110), “black, white, black” (101), and “white, white, black” (001) are mapped into a black cell (1), and the other neighborhoods are mapped into a white cell (0). Of course the 0s and 1s in the second row (the new cell states) could have been assigned differently, and there are $2^8 = 256$ ways of constructing an ECA update rule (since each of the 8 neighborhoods can be mapped into a 0 or a 1). The figure below shows a simple example of how cells in a CA lattice change their states given the above update rule. For example, the second cell in the lattice has a local neighborhood configuration of “white, black, black” (011), and according to the given rule will be white (0) at the next time step (first arrow). The fifth cell has a local neighborhood configuration of “white, white, black” (001), and thus will be black (1) at the next time step (second arrow). Similarly, all cells are changed to their new state simultaneously according to the same update rule given above.



In case of a finite lattice, as in the example above, we also need to specify boundary conditions. In this case, *periodic boundary conditions* are used, i.e., the lattice is considered to be circular. Thus, the first and last cells in this array are each other’s neighbor, and therefore the first cell has a local neighborhood configuration of “white (last cell), white (cell itself), black (second cell)” (001), and will become black (1) at the next time step according to the given update rule. Alternatively, fixed boundary conditions can be used, where an additional cell is placed on either side of the lattice with a fixed cell state (i.e., these boundary cells are not updated).

Depending on which one of the 256 ECA update rules is used, different behaviors (or dynamics) can be observed when iterating the CA. The figure below gives examples of so-called *space-time diagrams* for four different ECA. In these diagrams, the CA lattice is shown horizontally, while time is going down the page. In other words, the first row in each diagram is the CA lattice at the initial time step *t=0* (in these examples, the CA lattices are initialized randomly, i.e., for each cell a state, black or white, is chosen at random). The second row is the updated lattice at time step *t=1,* and so on for each next row. These space-time diagrams show 100 cells across (with periodic boundary conditions), for 100 time steps. Clearly, the four different update rules give rise to very different kinds of dynamics.



Computation in Cellular Automata

It is possible to use the patterns that form in these cellular automata dynamics to perform computations. Of course not all CA, or all patterns they produce, can be said to perform computations, but sometimes it is possible to construct a specific CA update rule to perform a certain computational task. Some examples are given next.

Formal language recognition

Smith used CA as formal language recognizers [1]. In particular, one-dimensional, radius 1 "bounded" CA were studied, i.e., CA with two special (fixed state) boundary cells on either end of the lattice. Smith proved that the class of languages that can be recognized by such CA is the class of context-sensitive languages. Some specific examples he included are the context-free language of palindromes (i.e., words that are the same when read from left-to-right and from right-to-left), and the context-sensitive language $\{a^m b^m c^m \mid m \geq 1\}$. For both cases, he showed the construction of a bounded CA that recognizes one of these languages in linear time. An input word, such as *aabbcc,* is given as the initial configuration to the CA, and it will then decide in linear time whether the input belongs to the language or not. The decision is made by using one particular cell as the “answer cell”. If an input word of length *n* belongs to the language, this particular cell will be in the “accepting” state after *n* time steps (iterations), otherwise it will be in the “rejecting” state. It is possible for the CA to make this decision in linear time, because it can process the symbols in the input word in parallel.

Arithmetic
In [2], one-dimensional “filter automata” were used to perform arithmetic. The difference between filter automata (FA) and regular CA is that in FA the cells are updated from left to right, instead of simultaneously. So, when updating a particular cell, the new cell states of neighbors to the left and current cell states of neighbors to the right are considered for the local neighborhood configuration. In this class of automata, often “soliton-like” structures (patterns) can be observed, i.e., propagating periodic structures that can pass through each other without destroying each other, but only shifting each other’s phase. Using these types of structures, in [2] a simple “adder” was constructed. Two (binary) numbers are given as input in the initial configuration of the FA, and after a certain number of iterations the lattice will contain the sum of these two numbers. Similar arithmetic operations (such as multiplication) can also be implemented in an FA this way.

Random number generation
Wolfram gives an example of an ECA that can be used as a pseudo random number generator [3]. For this particular ECA, the state values that are attained by one particular cell in the lattice seem to form a stream of bit values that is indistinguishable from a random stream. In other words, for a large enough lattice, one particular cell in this CA can function as a generator of (pseudo) random bit values. Since the description of an ECA is very simple and concise, this can of course provide many useful practical applications, for example in the area of cryptography, as suggested by Wolfram.

Image processing
Another, perhaps more practical, task that can be performed with CA is that of image processing. Suppose an image is given as input to a 2D CA, where each cell in the lattice corresponds to a pixel in the image. In other words, for black and white images, if a pixel in the image is white, the corresponding cell in the CA is initialized to white, and similarly for black pixels. For gray-scale images, we can use for example 256 states in the CA, each state corresponding to a particular gray-scale level. In [4], two CA update rules are given for two different image processing tasks: noise filtering and edge detection. It is then shown that using these CA rules, it is indeed possible to remove noise from, or to detect edges in, an given image.

Universal Computation in Cellular Automata

The examples above show how CA can be used for very specific computational tasks. However, it has been proved that, in principle, CA can perform *any* computational task. In other words, they are capable of universal computation. One relatively straightforward way of proving this is by showing that a CA can, in principle, simulate any given Turing machine. This was done for two-dimensional CA in [5]. Obviously, from this it follows that it is possible to construct a 2D CA that simulates a universal Turing machine as well.

However, the actual construction of such a CA, including the correct initial configuration, would be very complicated. Therefore, the next natural question is: “What is the simplest CA that can perform universal computation?” In [6], a one-dimensional, 7-state, radius 1 CA was constructed that can simulate a Turing machine. And finally, after an early conjecture by Wolfram, it was even proved that there is one elementary CA (known as rule 110) that is capable of universal computation [7]. So, even the simplest version of a CA is, in principle, computationally as powerful as any other computing device!

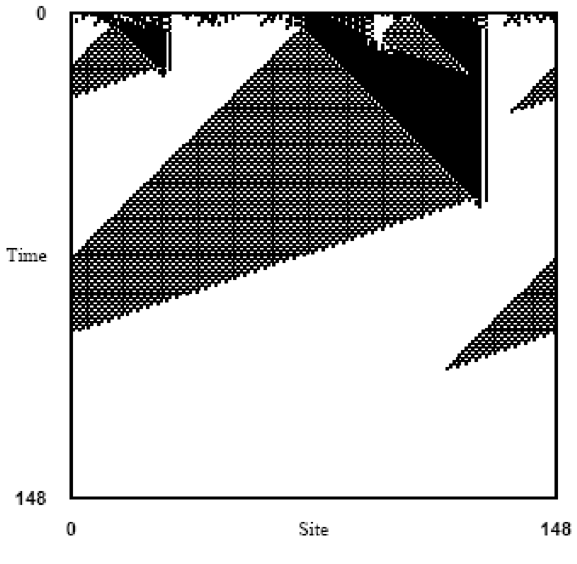
One early, but entirely different, proof of universality in CA was provided using a famous two-dimensional CA known as the “Game of Life”. This CA was originally invented by Conway, and is described in [8]. In this CA, there are only two possible states: “dead” and “alive”. The update rule is also simple. If a cell is alive, it remains alive if exactly 2 or 3 of its neighbors are alive, otherwise it dies. If a cell is dead, it remains dead unless exactly 3 of its neighbors are alive, in which case it becomes alive. The neighborhood of a cell consists of the 8 cells directly surrounding it (in a 2D lattice). However, despite this simple rule, the CA shows complicated behaviors and intricate structures, including propagating structures called “gliders”. These gliders are small (consisting of 5 alive cells) periodic structures, with a periodicity of 4 time steps, which are displaced one cell horizontally and vertically after one period. So, over time, these structures are propagated through the lattice, and interact with each other upon collision, either annihilating one or more of them, or changing direction, etc. Furthermore, there exist other structures that produce gliders (“glider guns”), destroy them (“eaters”), delay them (“delayers”), or make them change direction. As it turns out, these gliders and other structures can be used in such a way as to simulate any logical function, in particular AND, OR, and NOT gates. Since any logical system containing such functions is universal, it follows that the “Game of Life” is also universal, i.e., capable of universal computation.

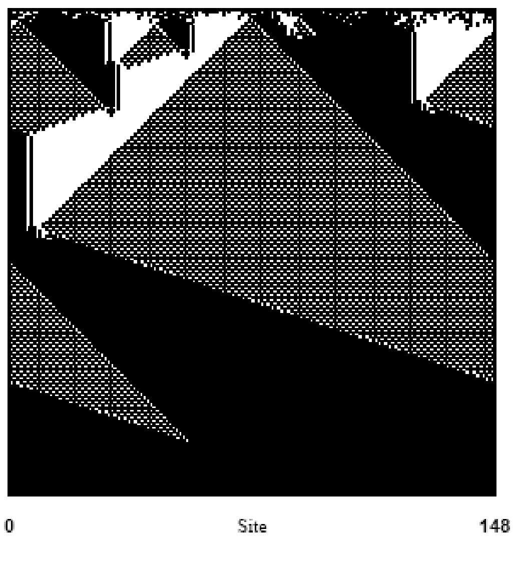
Emergent Computation in Cellular Automata

In the examples given above of computation in CA (specific or universal), a particular CA update rule and a corresponding initial configuration were constructed, or at least it was shown that such a construction is possible in principle. However, for most computational tasks of interest, it is very difficult to construct a CA update rule or it is very time consuming to set up the correct initial configuration. A very different approach was taken by Mitchell *et al.* in [9-11], where a genetic algorithm was used to *evolve* a CA rule to perform a specific computational task.

A *genetic algorithm* (GA) is a search algorithm based on the principles of natural evolution (see the introduction to evolutionary computation in this volume). It can be used to search through the space of possible CA rules to try to find a CA that can perform a given computational task. The specific task that Mitchell *et al.* originally considered is *density classification*. Consider one-dimensional, two-state CA. For any initial configuration (IC) of black and white cells, we can ask: "Are there more black cells or more white cells in the IC?" Note that for most "computing systems" (including humans), this is an easy question to answer, as the number of black and white cells can easily be counted and compared. However, for a CA this is a difficult task, since each individual cell in the lattice can only check the states of its direct neighbors, and none of the cells can have information of the global state of the CA lattice. So, the question is whether there exists a CA update rule that can decide on the density of black cells (smaller or larger than 0.5) for any IC. In particular, if a given IC contains more black cells (density >0.5), then the CA should settle down (within a certain maximum number of iterations) to a configuration of all black cells and remain in that state. Otherwise (density <0.5), it should settle down to all white cells.

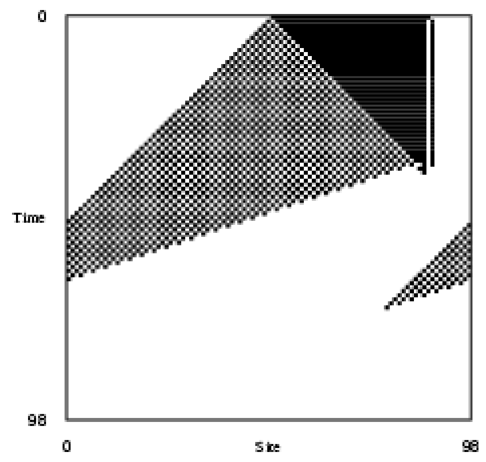
A genetic algorithm was used to search through the space of all one-dimensional, two-state, radius 3 CA update rules (of which there are $2^{128} \approx 3.4\times10^{38}$, i.e., too many to perform an exhaustive search). The way the different CA were evaluated during the search is as follows. A set of 100 random ICs is generated, and the CA being evaluated is iterated for *M* time steps on each of these. The lattice size was chosen to be *L*=149, and *M* was set to roughly 2*L*. The fraction of ICs on which the CA gives the correct answer (i.e., it correctly settles down to all whites or all blacks) is taken as the "fitness" of the CA, a measure of how well it performs the density classification task. Many runs of the genetic algorithm were performed, and in the end the best CA (the one with the highest fitness value) over all these runs was taken as "the solution". Two space-time diagrams of this "best" density classification CA are shown below. On the left the (random) IC contains more white cells, and on the right it contains more black cells. However, in both cases the CA correctly classifies the density of black cells. (The gray areas are alternating

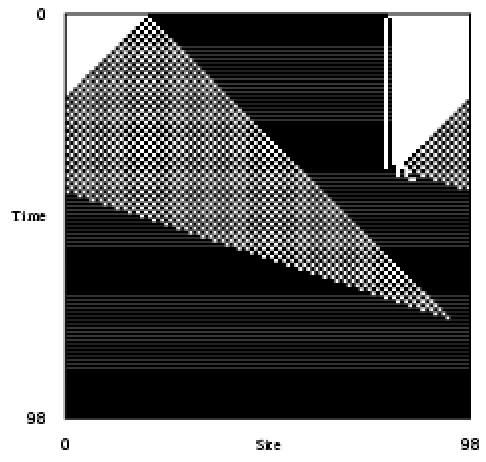




black and white cells, or a “checkerboard” pattern.) Periodic boundary conditions are used.
So how does this CA actually perform the density classification task? As is obvious from the space-time diagrams, the CA dynamics very quickly (after just a few initial time steps) settles down into a collection of very regular patterns, in particular regions of all white, all black, or checkerboard, with sharp boundaries between these regular regions. Furthermore, these boundaries move through the lattice at a certain velocity, and they interact with each other, creating new or destroying existing regular patterns and boundaries. So, it seems that the CA is using these patterns that *emerge* in its space-time dynamics to store, transfer, and process local information (local densities), eventually coming to a global decision about the overall density of the IC.

To appreciate the information processing, or computation, in this CA better, consider the two space-time diagrams shown below. In both cases, the IC contains one “block” of consecutive white cells, and one “block” of consecutive black cells (since periodic boundary conditions are used, the white region wraps around and thus forms one continuous block). At the boundary between the white (W) region and the black (B) region, a checkerboard (#) pattern is formed, and two boundaries are created (W# and #B), which travel in opposite directions but at the same velocity. At the boundary between the black and white regions, this boundary (BW) persists and remains in the same location over time (i.e., the vertical boundary). In the figure on the left, the original white block in the IC is larger than the black block, and therefore the #B boundary reaches the BW (vertical) boundary well before the W# boundary does. When the #B and BW boundaries meet, the black region and the two boundaries themselves are destroyed, and a new boundary (#W) is created that travels back in the same direction as the original W# boundary, but with a higher velocity. So, eventually the new #W boundary catches up with the existing W# boundary, and upon collision they annihilate each other and the checkerboard region, leaving the entire CA lattice in an all white state. As can been seen in the figure on the right, exactly the opposite happens when the white block in the IC is smaller than the black block. So, in both cases the density of black cells in the IC is classified correctly by the CA (<0.5 on the left and >0.5 on the right, with the correct corresponding answers of all white and all black, respectively).





Going back to the first pair of space-time diagrams where random ICs were used, it can be seen that this "strategy" of creating checkerboard patterns and using interacting boundaries, as just explained, is used at different time- and length-scales. Local information about densities in small parts of the lattice is stored in patterns of all white, all black, or checkerboard, and the boundaries between these regions transfer this information to other parts of the lattice, where the information is processed through interactions of these boundaries. Decisions about local densities are made this way, and are propagated to other parts of the lattice, until finally a global decision about the overall density of the IC is made. Since the CA makes use of these patterns that (spontaneously) emerge in its dynamical behavior, this type of information processing is generally referred to as *emergent computation* [12-14].

The above explanation of emergent computation in the evolved CA is a rather informal argument. However, the mechanisms of emergent computation in CA have been studied more thoroughly and have been formalized in a mathematical framework in [15,16], where a concise model has been developed based on the notion of these propagating and interacting boundaries. In fact, this model (which is a higher-level description of the dynamics of the CA) can be used to make accurate predictions about the CA's behavior and computational performance. Furthermore, it can be used to better understand the actual evolution (by the genetic algorithm) of emergent computation in CA.

More Information

Cellular automata were originally introduced by von Neumann after a suggestion by Ulam [17,18]. They were made popular by a series of articles by Gardner on the "Game of Life" in Scientific American (see [19] for an overview), but were more or less neglected otherwise. In the 80s, with the increase in computing power, research on CA revived again, and Wolfram was one of its pioneers (see [20] for a collection of his papers on CA). Since then CA have been used in many different areas and as models of many different systems. For a more elaborate overview of computation in cellular automata, which is only briefly presented here, see the chapter on this topic in [21]. And finally, many CA resources are available on the internet. A good starting point is `cafaq.com`, and any websearch on the topic will result in numerous additional pages. For more information on evolving cellular automata with genetic algorithms and emergent computation in CA, with links to downloadable electronic versions of publications including [9-13,15,16], please visit `www.santafe.edu/projects/evca`.

References

[1] A. R. Smith, "Real-time language recognition by one-dimensional cellular automata", *Journal of Computer and System Sciences*, vol. 6, pp. 233-253, 1972.

[2] K. Steiglitz, I. Kamal, and A. Watson, "Embedding computation in one-dimensional automata by phase-coding solitons", *IEEE Transactions on Computers*, vol. 37, pp. 138-145, 1988.

[3] S. Wolfram, “Random sequence generation by cellular automata”, *Advances in Applied Mathematics*, vol. 7, pp. 123-169, 1986.
[4] A. Popovici and D. Popovici, “Cellular automata in image processing”, *Proceedings of the 15th International Symposium on the Mathematical Theory of Networks and Systems*, 2002.

[5] A. R. Smith, “Simple computation-universal cellular spaces”, *Journal of the Association for Computing Machinery*, vol. 18, pp. 339-353, 1971.

[6] K. Lindgren and M. G. Nordahl, “Universal computation in simple one-dimensional cellular automata”, *Complex Systems*, vol. 4, pp. 299-318, 1990.

[7] M. Cook, “Universality in elementary cellular automata”, Complex Systems, vol. 15, pp. 1-40, 2004.

[8] E. Berlekamp, J. H. Conway, and R. Guy, “*Winning Ways for your Mathematical Plays*”, volume 2, Academic Press, 1982.

[9] M. Mitchell, P. T. Hraber, and J. P. Crutchfield, “Revisiting the edge of chaos: Evolving cellular automata to perform computations”, *Complex Systems*, vol. 7, pp. 89-130, 1993.

[10] M. Mitchell, J. P. Crutchfield, and P. T. Hraber, “Evolving cellular automata to perform computations: Mechanisms and impediments”, *Physica D*, vol. 75, pp. 361-391, 1994.

[11] R. Das, M. Mitchell, and J. P. Crutchfield, “A genetic algorithm discovers particle-based computation in cellular automata”, Parallel Problem Solving from Nature – PPSN III, pp. 244-253, 1994.

[12] J. P. Crutchfield and M. Mitchell, “The evolution of emergent computation”, *Proceedings of the National Academy of Sciences*, USA, vol. 92, no. 23, p. 10742, 1995.

[13] M. Mitchell, J. P. Crutchfield, and R. Das, “Evolving cellular automata with genetic algorithms: A review of recent work”, *Proceedings of the First International Conference on Evolutionary Computation and its Applications*, Moscow, Russia, 1996.

[14] S. Forrest (editor), “*Emergent Computation*”, North-Holland, 1990 (a special issue of *Physica D*, vol. 42, nos. 1-3).

[15] W. Hordijk, J. P. Crutchfield, and M. Mitchell, “Embedded-particle computation in evolved cellular automata”, *PhysComp96*, pp. 153-158, 1996.

[16] W. Hordijk, J. P. Crutchfield, and M. Mitchell, "Mechanisms of emergent computation in cellular automata", *Parallel Problem Solving from Nature*, pp. 613-622, 1998.
[17] J. von Neumann, "*Theory of Self-Reproducing Automata*", University of Illinois Press, 1966 (edited and completed by A. W. Burks).

[18] A. W. Burks (editor), "*Essays on Cellular Automata*", University of Illinois Press, 1970.

[19] M. Gardner,"*Wheels, Life and other Mathematical Amusements*", W. H. Freeman and Company, 1983.

[20] S. Wolfram, "*Cellular Automata and Complexity*", Addison-Wesley, 1994.

[21] T. Gramss, S. Bornholdt, M. Gross, M. Mitchell, and T. Pellizzari, "*Nonstandard Computation*", VCH Verlagsgesellschaft, 1998.