Escuela Superior de Cómputo
Instituto Politécnico Nacional
Ingeniería en Sistemas Computacionales

Evolutionary Computing

Practice 9: Cellular Automata

Professor: Jorge Luis Rosas Trigueros
Student: Ayala Segoviano Donaldo Horacio
Creation date: November 20 2021
Delivery date: November 23 2021

1 Introduction

1.1 Cellular Automaton

Cellular automaton are simple mathematical idealizations of natural systems. They consist of a lattice of discrete identical sites, each site taking on a finite set of, say, integer values. The values of the sites evolve in discrete time steps according to deterministic rules that specify the value of each site in terms of the values of neighboring sites.

Even though a formal definition has not been yet defined among cellular automaton resarchers community, the following is a definition:

A cellular automaton is a 4-tuple $A = (S, N, Q, \delta)$ where:

- S is a cell space.
- N is a neighborhood.
- Q is a set of states.
- δ is a function $\delta : Q^N \to Q$.

One of the simplest types of cellular automaton is the 1D cellular automaton, in which the space consists only of a 1D grid or array, and therefore, the neighborhood can only consider cells to the left and to the right of one cell.

1.1.1 Elementary Cellular Automaton

A well known example of 1D automaton is the Wolfram's Elementary CA, in which rules consider a neighborhood of only three cells (the nearest two and the cell itself) and only two states. It is one of the simplest automatons because there are only 8 possible states for the three. combinations of cells, these combinations can be mapped to calculate the value of the middle cell, making it easy to define rules.

Wolfram's cellular automatons are represented with the following notation Wx where $0 \leq x \leq 255$ which are the possible combinations of rules for the 8 states. The image 1.1 represents the cellular automaton for the rule $W30$.

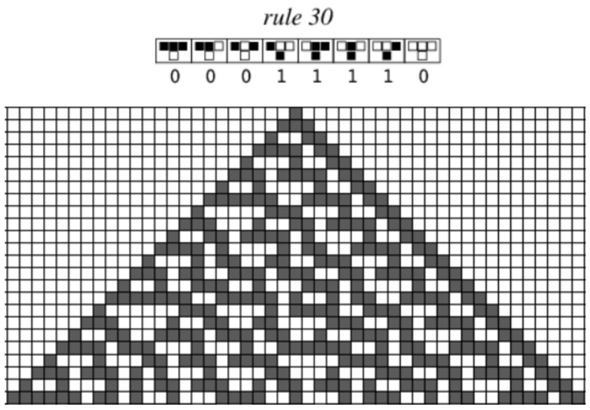

Figure 1.1: Visualization of the CA $W30$

1.1.2 2D Cellular automaton

Another type of cellular automaton is the 2D CA, in which the space consists of a 2D infinite grid of cells. In this case, there are different options to consider the neighbors, some of the most common are the the Moore neighborhood (figure 1.2 (a)) and Von Neumann neighborhood (figure 1.2 (b)). In this, the rules can also vary, usually the rules are determined according to the number of neighbors in a specific.

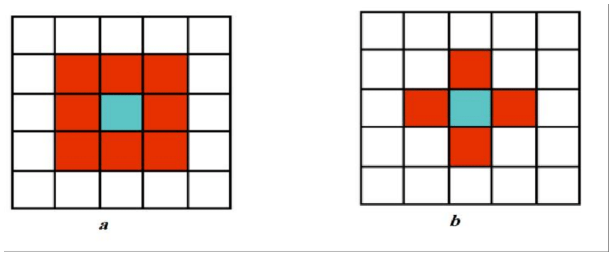

Figure 1.2: a) Moore Neighborhood. b) Von Neumann neighborhood.

An example of this automatons is the one considering only its closest neighbors (with Moore's neighborhood) and only has two states, it is represented with the following notation $BwxSyz$ in wich B stands for *born*, S stands for *survival* and w, x is the range of number of cells in state 1 for a new cell to be born and y, z is the range of number of cells in state 1 for the cell to remain in state 1. The figure 1.3 shows the evolution of a CA with the rules $B2S3$.

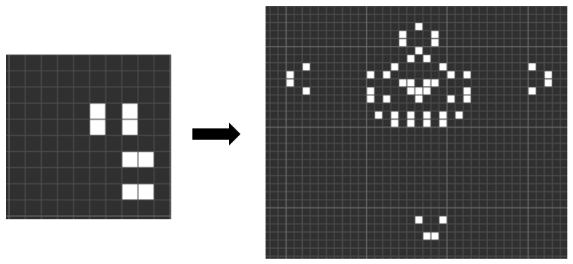

Figure 1.3: Visualization of the CA $B2S3$

1.2 Second Order Cellular Automaton

Second order cellular automatons are very similar to simple cellular automaton, but, second order CA's consider different rules when calculating the next state, the state of a cell of the second-order automaton at time $t+1$ is the exclusive or of its state at time $t1$ with the state that the ordinary cellular automaton rule would compute for it at time t.

The formal definition of a second order d the following:

- S = Infinite 1-D grid.
- N = {Closest neighbors} U { self }
- Q ={0,1}
- δ = The state of a cell of the second-order automaton at time $t+1$ is the exclusive or of its state at time $t1$ with the state that the ordinary cellular automaton rule would compute for it at time t.

And with this automaton, the concept of **reversibility** surges. The reversibility of an automaton is the capability of being able to determine past iterations, knowing only the last two states of the automaton, that is, the automaton can be reversed to its original state.

An automaton that has the reversibility characteristic is represented by adding a R at the end of the original representation. For example, if we apply the same rules as in $W30$ but with the second order modification, i.e. $W30R$, the result is the shown in the figure 1.4

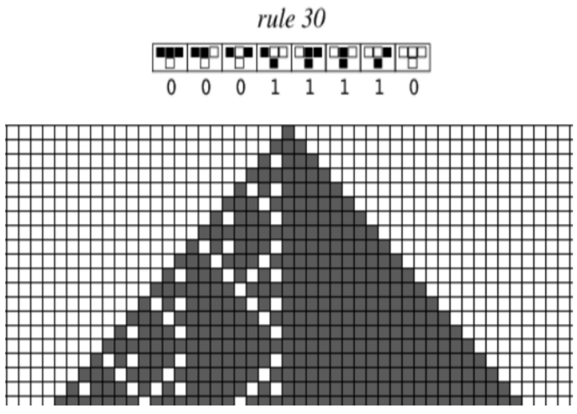

Figure 1.4: Visualization of the second order CA $W30R$

2 Material and equipment

Following are the hardware and software used during the realization of this practice. For this practice, new tools were used since it these tools provide a good development environment for graphical representations. Specifically, *Visual Studio Code* was used as a code editor and my personal computer was used to run the Python scripts. And the following are the specifications of the hardware used for the development of the practice:

- **Hardware:**
  - **CPU:** 11th Gen Intel(R) Core(TM) i7-11370H @ 3.30GHz.
  - **Memory:** 8GB
  - **Disk:** 74GB of free space
- **Software:**
  - **Code editor:** *Visual Studio* was used for this practice
  - **Programming language:** Python 3.9.6
  - **Modules used:**
    - pygame 2.1.0

3 Development

In this practice, three types of CA will be explored, first, given some specific rules for a 2D CA with three states and the Moore closest neighbors, the objective is to find rules and a configuration that generates either a glider, an oscillator or a still life that visits all three states. After that, a 1D CA will be explored and visualized for a specific configuration and a neighborhood of 5 cells (the cell itself, 2 cells to the right and 2 to the left).

3.1 Environment to execute cellular automatons

There are several software options available for the exploration of cellular automatons. However, since the configurations for this practice are quite specific, it was decided to create a program that shows the visualization of CA's with our specific needs. For this, the *pygame* library was used to visualize the automatons.

The script will create one of the three types of CA mentioned in the description, and it will pass the CA to a painter class which will show the state of the automaton in screen. Once implemented, the script will be used for the CA's of the next sections. The script of the CA's and the painter class can be found by clicking **here**.

The interface will allow to set the states of the cells by clicking on them, you can *right click* to set the state of the cell to 2, *left click* to set the state to 1, to start or stop the iterations you have to press *space bar*, and to reset the grid to 0, press *R*.

3.2 2D Cellular Automaton

For this section, the following rules will be considered:
A 2D $AC = (S, N, Q, \delta)$

where:

- S = infinite rectangular grid.
- N = {closest neighbors} U {self}
- $Q = \{0, 1, 2\}$
- $\delta = (b_1, b_2, s_1, s_2, u_1, u_2, r_1, r_2)$
- If the cell is in state 0, and the number of closest neighbors in state 1 is between b_1 and b_2 (b stands for born) the state of the cell will change to 1.
- If the cell is in state 1,

  - If the number of closest neighbors in state 1 is between s_1 and s_2 (s stands for survival) the state of the cell will remain in 1.
  - Else: If the number of closest neighbors in state 2 is between u_1 and u_2 (u stands for upgrade) the state of the cell will change to 2.

- If the cell is in state 2,
  - If the number of closest neighbors in state 2 is between r_1 and r_2 (r stands for remain), the state of the cell will remain in 2.
  - Otherwise, the state of the cell will change to 0.

Find either an oscillator, a still life or a glider that visits all three states and uses in δ at least two digits in your student number.

3.2.1 Exploration of rules and configurations

The first approach to find a configuration that meets such requirements was to test with values believed to be good for potential interesting evolution. However, it was quite difficult to change a configuration this was, and it was taking too long. So, in order to explore different solutions more efficiently, a mechanism to generate new configurations was implemented, in which by pressing a key, it would create a random new configuration and new rules.

This approach was very effective to explore different interesting CA's. After many attempts, a glider was spotted in a random configuration, and this random configuration contained two of my student number, so this CA was selected.

The CA has the following definition:

$AC = (S, N, Q, \delta)$
where:

- S = infinite rectangular grid.
- N = {closest neighbors} U {self}
- $Q = \{0, 1, 2\}$
- $\delta = (b_1 = 2, b_2 = 2, s_1 = 4, s_2 = 8, u_1 = 1, u_2 = 2, r_1 = 6, r_2 = 7)$

The create a glider, it is enough to place two state 1 cells together and two state 2 cells right next to the state 1 cells. The figure 3.2 shows some configurations of gliders that can be created in the automaton.

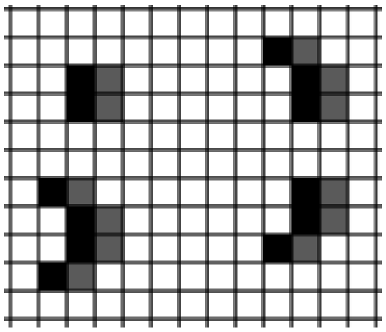

Figure 3.1: Variations of glider configurations for the CA.

An interesting behavior of this automaton was the fact that state 1 (gray cells) cells start spreading through all of the grid easily, but when there are state 2 cells (black cells) the state 2 cells start consuming the state 1 cells, and the CA enters in a cycle with some kind of chaos. Figure **??** shows two different moments of the evolution of the CA, it can be seen that even if black cells consume the gray ones, the gray ones will always spread again through the grid.

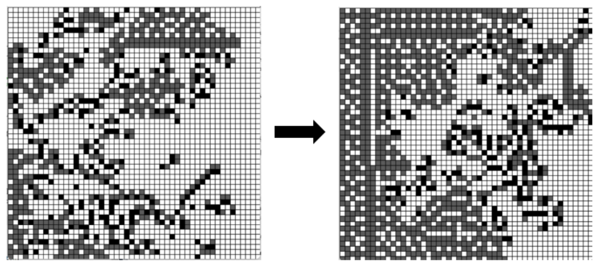

Figure 3.2: CA evolution.

3.3 1D Cellular Automaton

For this section, the following specifications will be followed:

For a 1D $CA = (S, N, Q, \delta)$

where:

- S= Infinite 1-D grid.
- N= {First 2 layers of closest neighbors} U { self } ($|N| = 5$)
- Q={0,1}

Provide a graphical representation of the evolution of this CA with:

- $\delta = Wxy0$ and
- $\delta = Wxy0R$

Where x, y are nonzero digits from your student number. Provide evolutions that start with a small number of cells in 1 and with random configuration.

3.3.1 Common 1D cellular automaton

For this CA, it was decided to use my own student number, and the only modification made to it, was to change the last digit for a zero, to avoid weird behaviors, so, the rules where the number 2019630410 in binary, and after converting the student number to binary, one zero was inserted at the beginning to get the 32 rules for the CA.

The image 3.3 shows the evolution of the CA for only one cell in state 1 in the first iteration. It can be observed that some triangles were formed, similar to the Sierpinski triangle, but inverted, and in this case the triangles are deformed.

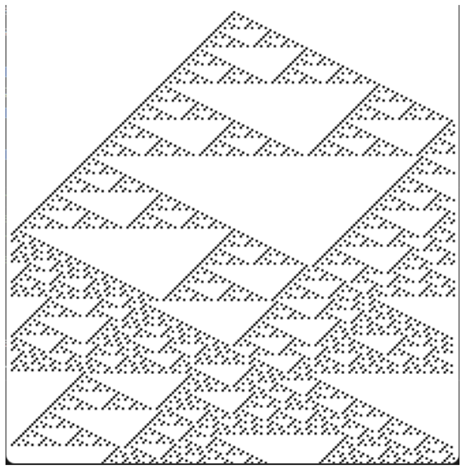

Figure 3.3: 1D Cellular Automaton $W2019630410$ for only 1 cell in state 1 in initial configuration.

Now, a random configuration will be tested with a small number of 0's in the initial state. Figure 3.4 shoes the output for the random initial configuration. It can be observed that the randomly generated initial state only has 4 cells in 1, and the result is the same pattern but repeated randomly across the grid, we can see the triangles of different sizes.

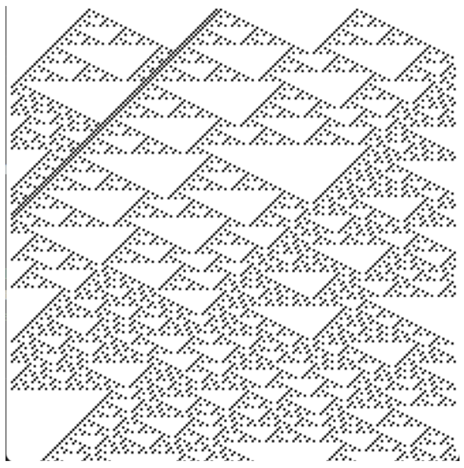

Figure 3.4: 1D Cellular Automaton $W2019630410$ for initial random configuration.

3.3.2 Second Order 1D Cellular Automaton

Now, the same rules will be provided to the second order 1D CA. First, it will be run with only one cell in the initial state set to 1, to see the kind of pattern generated. Figure 3.5 shows the pattern generated by only on cell set to 1 in the initial configuration, it can be seen, that at the left, some order is taken, whereas in the right, there is chaos in the states. Also, when the ordered pattern reaches the wall (because of implementation) the pattern changes to chaos just like in the right side.

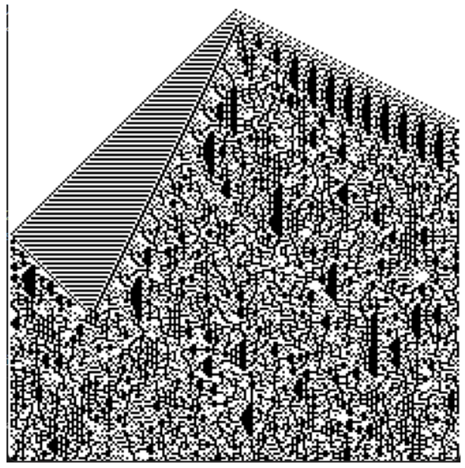

Figure 3.5: 1D Cellular Automaton $W2019630410R$ for 1 cell set to 1 in initial configuration.

Now, more cells will be set to 1 in the initial configuration, and the configuration will be set randomly. The figure 3.6 shows the pattern generated by this initial configuration. It can be observed that four cells were set, and that before combining, they all follow the same pattern as in the last run shown in figure 3.5, but when they merge, two of them generate a complete black triangle and the rest remains in chaos.

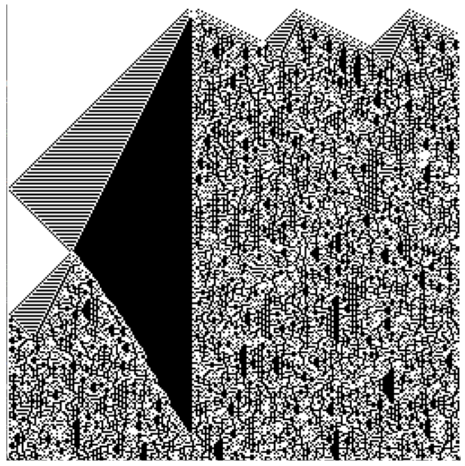

Figure 3.6: 1D Cellular Automaton $W2019630410R$ for random initial configuration.

4 Conclusions

The development of this practice resulted in a better and deeper understanding of cellular automatons, it was understood how they work and how they can be implemented. It was concluded that even when having very simple rules, they can act in very complex ways. It was also concluded that the implementation for a cellular automaton is relatively simple, since the rules are also simple and easy to represent in code.

One of the observations made during this practice, was that creating rules for a 2D cellular automatons that is interesting and has either an oscillator, a glider or a still life is quite difficult, since it is very hard to think of the configurations in advanced.

Also, it was concluded that cellular automatons can have very interesting behaviors, and some of them, can even be used to model real life physics or behaviors, such as the behavior found in the 2D CA, in which black cells try to consume gray cells, similar to immune system cells destroying bacteria.

5 Bibliography

- **Weisstein, Eric W. (2021) "Cellular Automaton." From MathWorld. A Wolfram Web Resource. https://mathworld.wolfram.com/CellularAutomaton.html.**

- **Wolfram Stephen. (1983) Cellular Automata. Los Alamos Science. USA.**

- **Trigueros Jorge. (2021) "Cellular Automata". Lecture. Escuela Superior de Cómputo - IPN.**

- **Weisstein, Eric W. (2021) "Elementary Cellular Automaton." From MathWorld–A Wolfram Web Resource. https://mathworld.wolfram.com/ElementaryCellularAutomaton.html**