Deterministic Modelling

Cellular Automata Sierpinski Triangle

Professor:
Jose Sardañes Cayuela
Authors:
Sergi Picó Cabiró 1531767
Sofia Almirante Castillo 1527718

1 Abstract

This work will consist of a deep analysis of the Rule90 algorithm, which generates patterns in Cellular Automata. In order to do so, we will create a python code that will simulate this algorithm for the 1-Dimensional case and the 2-Dimensional case. In addition, we will see how all the structures that emerge from this simulation change as we change the initial conditions and in the end we will change a little bit this algorithm so that it is applied with a certain probability. The analysis will be qualitative and extracted from the plots that we obtain from the simulation. All in all, we will extract conclusions on this particular algorithm and on Cellular Automatas in general.

2 Introduction

In this project, we will deal with a cellular automata model which generates the so-called Sierpinski triangle, a fractal structure with very rich patterns which outline the great capacity to obtain complicated patterns from very simple rules.

Cellular automata (CA) are mathematical models of complex systems that are composed of simple, identical units that update their state based on a set of rules according to the states of neighboring units. We can find arbitrary dimensional CA, and we can reach complex patterns in any of those dimensions. For this reason, CA are explained in the context of dynamical systems where the space, time and the variable itself (state space) is discrete. In this project, we will solve a 1-dimensional and 2-dimensional CA model.

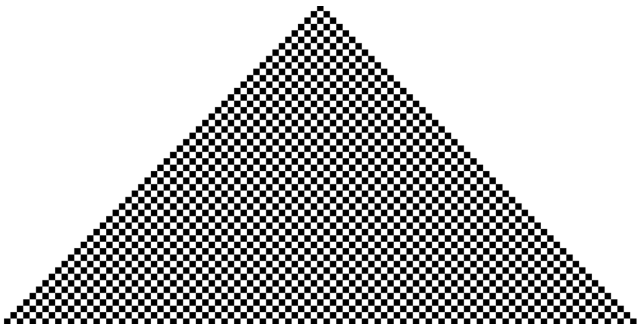

Figure 1: 1-Dimensional CA with periodic structures

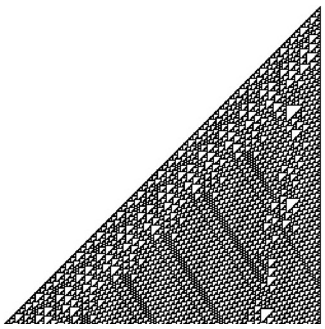

Figure 2: 1-Dimensional CA with complicated localized and propagating structures

Figures of some patterns created by 1-Dimensional CA yielding different types of structures

3 Model Description

Specifically, we will be analyzing the Rule 90 model. Suppose that we have a 1-Dimensional grid, where all the cells that form it can only be in two different states: 1 and 0 state. We consider that initially the system is found in a particular configuration with some cells initialized to 0 and other ones initialized to 1. The next configuration of the grid is determined by the rule that we will be using. This rule has information on the state of a particular cell on the next time iteration from the neighbor cells states. The Rule 90 is given by:

Table 1: Rule 90 explanation

State of cell and neighbors at t	000	001	010	100	101	011	110	111
State of cell at $t+1$	0	1	0	1	0	1	1	0

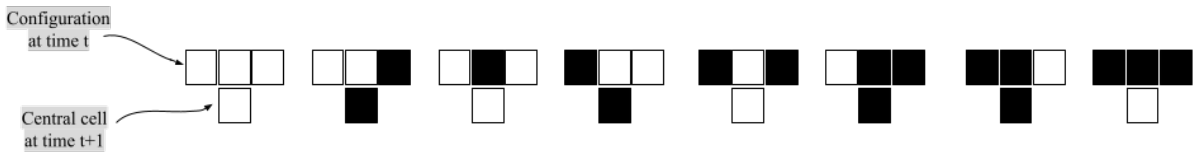

Figure 3: Visualization of the Rule 90 with actual cells and its neighbors

Where it is implemented the *exclusive or* logic gate which states that if the two neighbor cells are different from each other we will have a 1 outcome and a 0 otherwise.

All in all, we will study this system from two different frameworks: 1D and 2D. In any case we will proceed as follows:

1. Develop the code and study the general pattern that it is obtained from it.
2. Study how these patterns change when we associate a probability p that 3 particular cells give the described outcome.
3. Notice of the structures created when the initial conditions are changed.

4 1-Dimensional case

Let us analyze and study the evolution of this model from the 1-D framework. In this context, the total space is considered to be a single array formed by N units (cells) where each of them can be in state 0 (represented by a white cell) or 1 (represented by a black cell).

In each time iteration, all the array is actualized and the configuration of the state of the cells changes. So, we will be representing all the configurations in a single matrix, where each row represents the array at a certain time iteration and the columns represent the time going forward.

We have developed a code that simulates everything that we have mentioned, where the length of the array is $N = 51$ and we are repeating the process $M = 60$ time iterations.

1. In this case, we suppose that the initial condition is that the central cell of the array is 1 and the rest are in the 0 state. Once we simulate the rest of the configurations for all the time, we obtain the following plot:

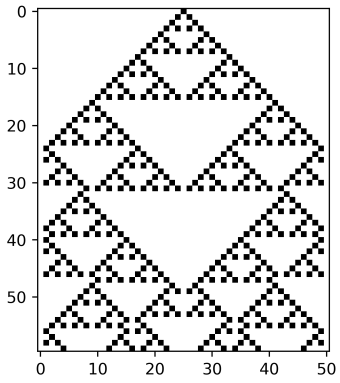

Figure 4: First simulation with $N = 51$, $M = 60$ and only the central cell initialized to 1.

Thus, the structure that this Rule90 yields is the so-called *Sierpinski Triangle* which it is a fractal since it has details at any scale. Moreover, it is self-similar at these different scales, and it is characterized by the triangular-equilateral shapes that form it.

2. Now we will modify a bit the rule and apply the Rule 90 to every cell with a probability p. The plot that we get with the same dimensions and initial condition as before is:

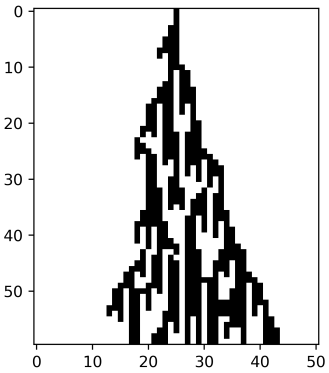

Figure 5: 2nd simulation with $p = 0.3$

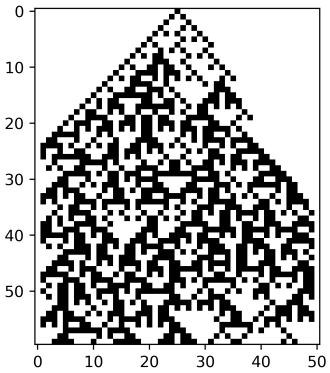

Figure 6: 2nd simulation with $p = 0.9$

As we can see, the figures that we obtain are relatively similar to the one obtained before but, in this case they look more stretched. In fact, the lower the probability, the more extended these structures appear. The reason why now we get this result is that a cellular automate will not apply necessarily the Rule90 at each time step. Specifically, it will do it with a certain probability p and there exists the possibility that after iterating a few times, the rule is not applied and a same cell does not change its state for a while. The lower this p is, the larger these columns might look.

3. Finally, we will make use of the first simulation (we are not applying the probability p case) but we will change the initial conditions a little. If we choose the initial condition to take a random configuration, we will obtain a plot of the form:

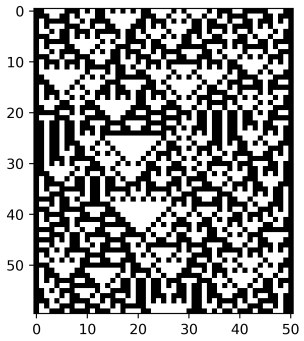

Figure 7: 3th simulation with random initialization

Where we observe a plot which is a combination of order and randomness. There are regions where we can detect repetitiveness and simple structures, but others look very chaotic. However, if the initial condition that we set is that all cells are at the same state we get:

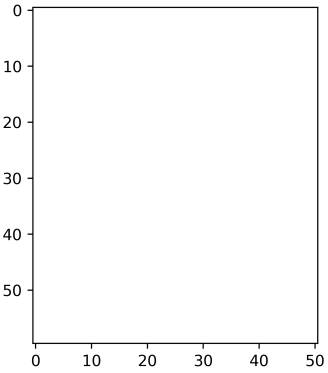

Figure 8: 3th simulation with 0 first row

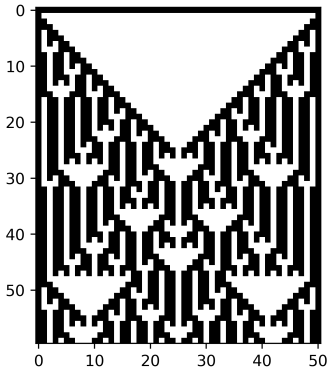

Figure 9: 3th simulation with 1 first row

In the first case, all the following rows that the algorithm yields are in the 0 state and we obtain a fully homogeneous matrix where all the cells are in the state 0. On the other hand, in the second case we obtain some regular patterns where we can see some triangular-shape structures, self-similarity and fractal characteristics. However, it is not the same pattern as the one observed in the Sierpinski triangle.

5 2-Dimensional case

For the 2-Dimensional case, we are going to extrapolate the previous rule as follows: If all the neighbors are the same, the central cell will be 0 (white). Otherwise, it will be 1 (black). For this reason, we are going to implement small variations in the 1D code (see Appendix).

We have implemented this rule in a von Neumann neighborhood, meaning we consider the center cell plus its 4 nearest neighbors, which gives a 5-cell neighborhood in the form of a cross. But it could also be possible to use the Moore neighborhood, considering the center plus its eight surrounding cells forming a square. Considered like this, the rule would return 1 to the central cell in neighborhoods 0 and 15 shown in Fig. 10 and 0 otherwise.

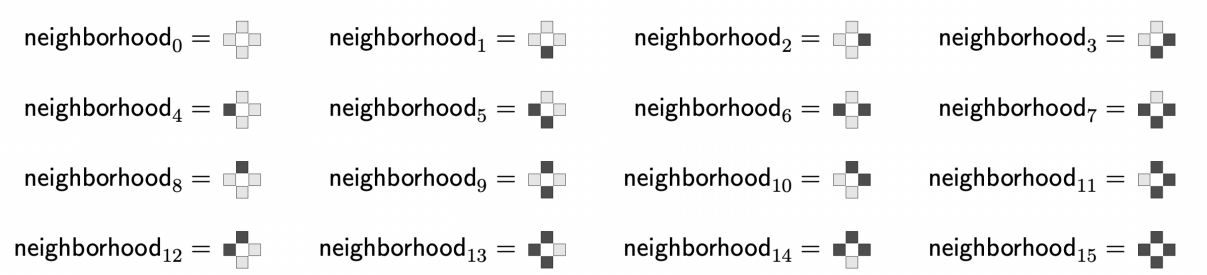

Figure 10: Possible neighborhoods in the 2D scenario.

We have considered a square matrix with $N \times N$ cells (and once again, we have chosen $N = 51$). In each time iteration, the whole matrix is actualized. For this reason, we will represent the time dynamics with different matrices representing different time steps, and providing animated GIFs.

1. In this case, we suppose that the central cell of the matrix is 1 and the rest are 0. We can see some steps of the evolution in the next figure:

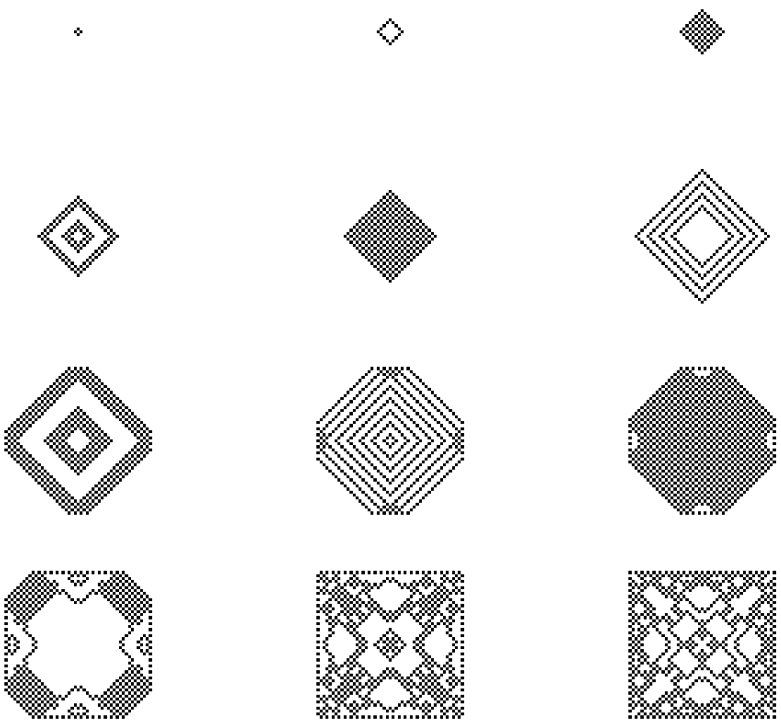

Figure 11: Time evolution of the matrix with the central cell initialized to 1.

Where we observe a symmetrical growing pattern that changes its dynamics when it hits the borders. In Fig. 12 there has been represented the central row of the matrix over time as we did in the 1D scenario. We can see that the pattern show is the same as the one produced by the 90 Rule. For this reason, we correctly justify using this 2D method as a generalization of the already explained 1D model.

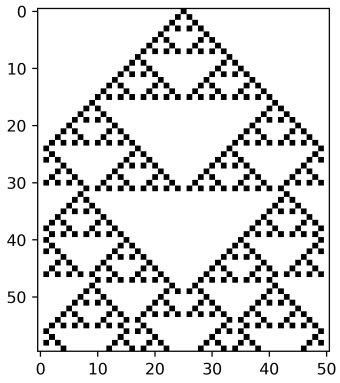

Figure 12: Time evolution of the central row when the central cell is initialized to 1.

2. Once again, as we did in the 1D case, we are going to modify the behavior of the system so the probability of change according to the rule is p. In Fig. 13 and 14, we have represented the time evolution of the matrix with $p = 0.5$ and $p = 0.8$ respectively. We can see the case with bigger probability resembles more to the original scenario, while a lower probability shows a more random and chaotic growth (completely equivalent to the 1D analysis).

Figure 13: Time evolution of the matrix with $p = 0.5$

Figure 14: Time evolution of the matrix with $p = 0.8$

In Fig. 15, we have represented over time the central line of the matrix of the previous scenarios. There is a similarity to what we observed in the 1D case conditioned to a probability, as it shows a stretched evolution with lower probability.

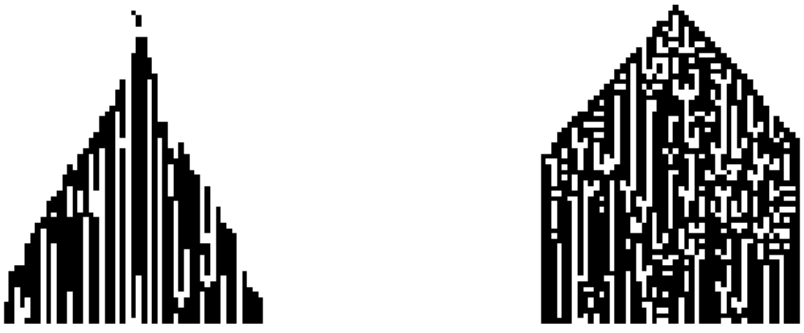

Figure 15: Time evolution of the central row for $p = 0.5$ (left) and $p = 0.8$ (right).

3. In a last place, we have studied the evolution of the system with different initial conditions. When initializing the matrix to 0, it shows the same behavior as in the 1D case, being homogeneous over time a matrix full of zeros.

However, in Fig. 16 we have represented the time evolution of the matrix when 1% of random cells have been initialized to 1. We observe the evolution we found in 11 in smaller cases, with a more complex and non-repetitive behavior once the structures are mixed. In Fig. 17 we have represented the same, but initializing a 10% of the matrix. The results show a more complex behavior from the beginning due to the higher interaction of the simpler structures.

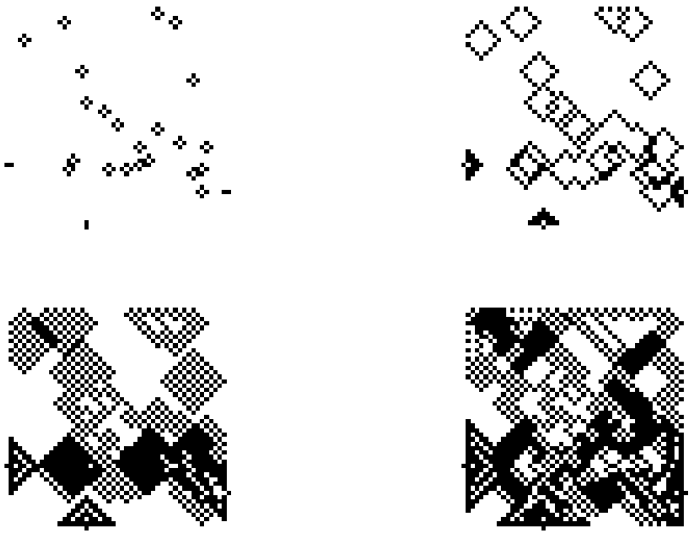

Figure 16: Time evolution of the matrix with a 1% of random cells initialized to 1.

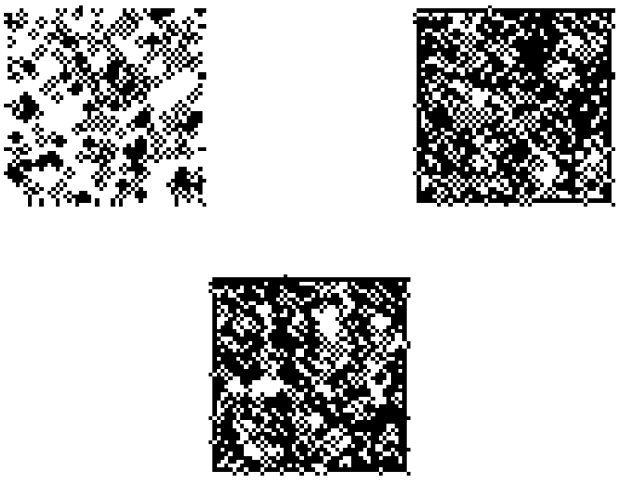

Figure 17: Time evolution of the matrix with a 10% of random cells initialized to 1.

Finally, we have simulated the time evolution of the CA when all the borders are initialized to 1. In the following figures, we observe some particular configurations corresponding to specific time iterations.

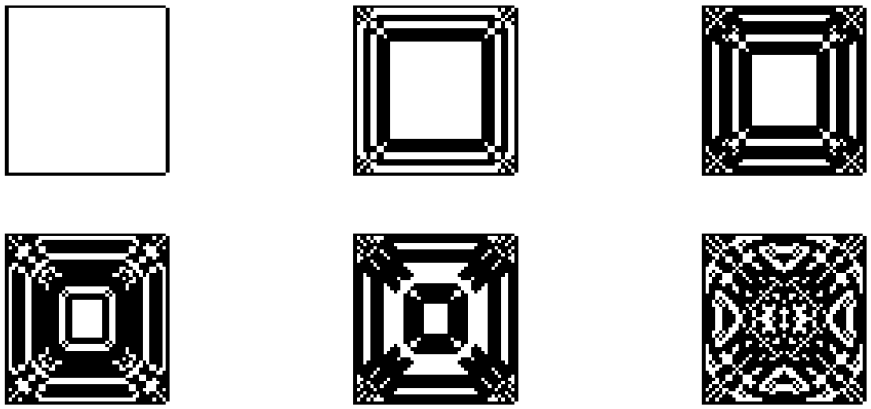

Figure 18: Time evolution of the matrix when initialized in the borders.

The structures that are formed again have a very rich behavior, with repetitive patterns and with symmetric structures. Just as happened for the 1D case, the patterns are not the same as the ones obtained when just the central cell was initialized. However, there is a feature of interest that is worth mentioning. All these figures have 4 symmetry axes (from top to bottom, left to right and the diagonals).

6 Conclusions

We have accomplished all the goals that we set at the beginning and all the analysis have been completed with success.

We have analyzed how repetitive, fractal and clear-structure patterns are accomplished for the easier case where no probability is taken into account and the initial condition is that the central cell is initialized to 1.

In addition, we have checked out the clear effect of stretching to the structures when we introduce a probability that the Rule90 is applied. We also have noticed of structures in this case, but not as clearly as in the previous case. In fact, the lower the probability of applying the algorithm, the more different and chaotic structures we have obtained.

And finally, we have checked out how we can obtain plenty of different patterns when the initial conditions are changed. If there are more cells initialized to 1, it is more likely to find chaotic structures which do not follow any pattern. We also mention the particular case where all the boundaries of the matrix are initialized to 1, where even though self-similar structures are not displayed, very rich and beautiful patterns are created.

All in all, we note that all these analyses have been common to the 1D case and the 2D case. That is why we conclude that we have made correctly the generalization from 1D to 2D. And last but not least, it is worth mentioning that we have imposed non-periodic fixed boundary conditions. In this case, all the cells that are found in the frontier, do not change its state, thus, structures are less likely to interact with each others when they are found close to the borders. We can study how patterns are changed if we impose periodic boundaries in future works, where we will clearly show how CA display very interesting and more complex structures.

7 Appendix

First code 1D

```
import numpy as np
import matplotlib.pyplot as plt
N=51 #Length of the array
M=60 #Time steps
grid=[]
###Initialization###
A=np.zeros(N)
A[int(N/2)]=1
Anew=A.copy()
###Exclusive or###
def exor(a,b):
    if a!=b:
        return 1
    if a==b:
        return 0
###Time iterations and implementation###
for k in range(M):
    for i in range(1,N-1):
            Anew[i]=exor(A[i-1],A[i+1])
    grid.append(A)
    A=Anew.copy()
###Plot###
plt.imshow(grid, cmap='gist_yarg')
plt.show
```

Second code 1D

```
###Time iterations and implementation###
for k in range(M):
    for i in range(1,N-1):
        r=random.random()
        if(r<p):
            Anew[i]=exor(A[i-1],A[i+1])
            grid.append(A)
    A=Anew.copy()
```

Third code 1D

```
###Initialization###
A=np.zeros(N)
for i in range(N):
    e=random.random()
    if(e<1):
        A[i]=1
Anew=A.copy()
```

First code 2D

```
import numpy as np
import matplotlib.pyplot as plt
import imageio

N=51 #Length of the array
M=100 #Time steps
grid=[]

###Initialization###
A=np.zeros((N,N))
A[int(N/2)][int(N/2)]=1
Anew=A.copy()

###Exclusive or###
def exor(a,b,c,d):
    if a==b and b==c and a==d:
        return 0
    else:
        return 1
###Time iterations and implementation###
image_list=[]
for k in range(M):
    for i in range(1,N-1):
        for j in range(1,N-1):
            Anew[i][j]=exor(A[i-1][j],A[i+1][j],A[i][j-1],A[i][j+1])
    A=Anew.copy()
    grid.append(A[int(N/2)])

    plt.imshow(Anew, cmap='gist_yarg')
    plt.savefig(f'{k}.png')
    image_list.append(imageio.imread(f'{k}.png'))

plt.imshow(grid, cmap='gray_r', vmin=0, vmax=1)
plt.axis('off')

imageio.mimsave('animation.gif', image_list, fps = 5)
```

Second code 2D

```
p = 0.8
###Time iterations and implementation###
for k in range(M):
    for i in range(1,N-1):
        for j in range(1,N-1):
            r=random.random()
            if(r<p):
                Anew[i][j]=exor(A[i-1],A[i+1])
    A=Anew.copy()
```

Third code 2D

```
p = 0.1 #probability
###Initialization###
A=np.zeros((N,NN))
for i in range(N):
    for j in range(NN):
        e=random.random()
        if(e<p):
        A[i][j]=1
Anew=A.copy()
```