A symbiosis between cellular automata and genetic algorithms

Umberto Cerruti, Simone Dutto, Nadir Murru

Università degli Studi di Torino, Department of Mathematics "G. Peano"

Abstract

Cellular automata are systems which use a rule to describe the evolution of a population in a discrete lattice, while genetic algorithms are procedures designed to find solutions to optimization problems inspired by the process of natural selection. In this paper, we introduce an original implementation of a cellular automaton whose rules use a fitness function to select for each cell the best mate to reproduce and a crossover operator to determine the resulting offspring. This new system, with a proper definition, can be both a cellular automaton and a genetic algorithm. We show that in our system the Conway's Game of Life can be easily implemented and, consequently, it is capable of universal computing. Moreover two generalizations of the Game of Life are created and also implemented with it. Finally, we use our system for studying and implementing the prisoner's dilemma and rock-paper-scissors games, showing very interesting behaviors and configurations (e.g., gliders) inside these games.

Keywords: Cellular Automata, Genetic Algorithms, Game of Life, Prisoner's Dilemma

1. Introduction

The advent of electronic computers allows to make simulations of every kind of environment and even create new form of life with behaviors similar to the real ones. With these instruments, it is possible to simulate and study situations difficult to examine in real life, or prevent future developments of current problems.

Cellular automata are one of the most used tool in these situations. They were introduced by Von Neumann in 1951 [29] to resolve a problem of self-

replicating automata and they quickly evolved in the following years (see [44] for a good overview). Their outbreak was in 1970, when Gardner popularized the renowned Game of Life invented by Conway [14]. Some recent studies about the Game of Life can be found in [7, 15, 13, 32, 43]. Cellular automata have been applied in several fields, like modeling of traffic [5, 10, 27], pattern recognition [17, 40], cryptography [28] and many others. Further recent studies about cellular automata can be found in [1, 6, 18, 19, 33, 34, 36].

A connected but different approach is given by genetic algorithms. They were pioneered in the 1960s by Holland [16], who wanted to study the phenomenon of adaptation and implement it into computer systems. He took inspiration by the natural selection concept introduced by Darwin. The obtained procedure became popular only in the late 1980s and now is applied particularly in computer science for optimization and search problems [8, 9, 23, 45].

Since structures of cellular automata and genetic algorithms are similar, they can be merged into a new kind of instrument that combines the advantages of both and can be used to find new interesting case studies. The idea to evolve a cellular automaton by means of evolutionary algorithms is not completely new. For instance, in [21], [30], [41], evolutionary cellular automata have been used for studying and describing strategies of the prisoner's dilemma game. In [3], the authors studied the use of genetic algorithms for finding rules of cellular automata such that they display a desired behavior focusing on the case of solving the majority problem with cellular automata. Also in [24] and [25], the authors exploited genetic algorithms for evolving cellular automata for specific computational tasks such as density classification and synchronization. Further studies can be found in [4], [12], [31], [37], [39].

In this paper, we describe our system, which we call *cellular evolution*, as a bi-dimensional cellular automaton which works with rules inspired by genetic algorithms. Our purpose is to explain its behavior, show its prospect, and make this new procedure available to everyone. The implementation in Python of our work is completely open source and available at:
https://github.com/duttos/cellular_evolution.

The new cellular automaton is described in detail in Section 2 where, after the theoretical formulation, the steps of an algorithm in pseudo-code are given. In Section 3, the classic Conway's Game of Life [14] is implemented with cellular evolution and also two different generalizations with more than two states are obtained. Finally, sections 4 and 5 are devoted to the imple-

mentation and discussion of the prisoner's dilemma and rock-paper-scissors games, respectively.

2. Cellular evolution

In this section we introduce our cellular automaton inspired by genetic algorithms.

2.1. A new idea and its implementation

While in simple cellular automata each cell evolves according to a rule based only on the states of cells in its neighborhood, genetic algorithms require a fitness function which allows each cell to select the best mate in a genetic pool, a crossover operator which defines couples interactions and the resulting evolution of cells states, and a final possible mutation.

2.1.1. Initialization

Our principal implementation uses bi-dimensional lattices $\mathcal{L}$ of squared cells, but it is possible to generalize the process to every kind of structure. It is also possible to choose a boundary condition (torus, horizontal or vertical cylinder, closed) and/or the presence of a fixed nodes grid.

The local value space Σ is a finite set of different values, not necessarily numeric, assumable by each cell. In cellular automata, we usually consider a set of numeric values like $\mathbb{Z}_k$. We adopt the notation $\sigma_{\underline{i}}(t)$ for the state of cell $\underline{i}$ at the instant t.

At the beginning of a run, each cell $\underline{i} \in \mathcal{L}$ assumes a value $\sigma_{\underline{i}}(0) \in \Sigma$ given as input or chosen randomly.

2.1.2. Rule of cellular evolution

As for genetic algorithms, the rule of cellular evolution consists in three steps: selection, crossover and mutation.

In order to obtain a cellular automaton, during computations for selection and crossover we consider limited neighborhoods instead of the mating pools of genetic algorithms. In particular, we use Moore neighborhoods but it is a completely indifferent choice.

Selection. For each cell a fitness value is evaluated, depending on the state of the cells in its neighborhood: given a cell $\underline{i} \in \mathcal{L}$ and its neighborhood $\mathcal{N}(\underline{i}) = \{\underline{j}_1, \underline{j}_2, \ldots, \underline{j}_9\}$ we call the *configuration* of $\underline{i}$ at time t:

$$\mathcal{C}_{\underline{i}}(t) = (\sigma_{\underline{j}_1}(t), \sigma_{\underline{j}_2}(t), \ldots, \sigma_{\underline{j}_9}(t)), \tag{1}$$

and we consider a *fitness function* that takes these configurations and returns values in an ordered set $\mathcal{F}$ containing possible fitness values

$$f : \Sigma^9 \longrightarrow \mathcal{F}\,, \quad f(\mathcal{C}_{\underline{i}}(t)) = f_{\underline{i}}(t). \tag{2}$$

In this way, at each time step, every cell receives a fitness value that depends on the states of the cells in its neighborhood. After that, each cell $\underline{i}$ selects randomly $\underline{j}$, one of the fittest cells in $\mathcal{N}(\underline{i})$, to interact with it. Hence fittest cells have higher chances of being chosen for couplings.

Crossover. Now every cell has a selected mate, and the coupling can start. Since we want to keep the system as generic as possible we consider the following operator that takes states and fitness values of the two cells, together with the maximum values for state and fitness, and returns the new state value of the cell $\underline{i}$:

$$\begin{gathered} \chi : \Sigma \times \mathcal{F} \times \Sigma \times \mathcal{F} \times \mathbb{N} \times \mathbb{N} \longrightarrow \Sigma\,, \\ \chi\big(\,\sigma_{\underline{i}}(t), f_{\underline{i}}(t), \sigma_{\underline{j}}(t), f_{\underline{j}}(t), \mathit{maxp}, \mathit{maxf}\big) = \sigma_{\underline{i}}(t+1)\,. \end{gathered} \tag{3}$$

Unlike what happens in genetic algorithms, the result of the crossover is only one offspring, which takes the place of the parent cell $\underline{i}$, otherwise the system would not be a cellular automaton.

Mutation. Finally, with a very small probability, the state of some cells is randomly changed.

The resulting process is a cellular automaton with rules inspired by genetic algorithms. We observe that, despite during each single step we consider Moore neighborhoods, the dynamic rule of our system seen as cellular automaton uses different neighborhoods: for the cell $\underline{i}$ it takes the states of the cells in

$$\overline{\mathcal{N}}(\underline{i}) = \bigcup_{\underline{j} \in \mathcal{N}(\underline{i})} \mathcal{N}(\underline{j}), \tag{4}$$

and returns the new state of $\underline{i}$. Hence the neighborhood used by the dynamic rule of cellular evolution is the one in Fig. 1.

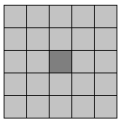


Figure 1: Neighbourhood used by cellular evolution as cellular automaton.

2.2. The algorithm step by step

Trying to keep the argumentation as generic as possible, now we use a pseudo-code to describe our methods of implementation.

2.2.1. Rules

Given an initial population *pop*, represented by a $m \times n$ matrix with entries in Σ, it evolves for N steps, in each of which the dynamical rule is applied as described in Section 2.1.2.

Selection. The fitness function is an operator `fitfun` which takes the state values of the nine cells in a neighborhood and returns a value in $\mathcal{F}$ as fitness value for the center cell. At each step, `fitfun` takes the population matrix *pop* and returns a matrix fit of dimensions $m \times n$ containing the fitness values of all the cells (line 3 in Algorithm 1).

Each cell selects the fittest mate in its neighborhood and, in case of cells with same fitness value, one of them is chosen randomly by exploiting an array called *order*. It is shuffled with a given *seed* at every step, allowing to sort randomly the cells in the neighborhoods (lines 1, 4 in Algorithm 1).

Before the selection, two three-dimensional matrices of size $9 \times m \times n$ are generated: *pp* and *ff*. They contain the values of state and fitness in the neighborhood of each cell, respectively. In particular, each vector $pp[\,[0, \ldots, 8\,], x, y\,]$ contains the states in the neighborhood of the cell (x, y) while each $ff[\,[0, \ldots, 8\,], x, y\,]$ contains their fitness values (in both cases in the random order defined by *order*. In the definition of these matrices in lines 5-6 of Algorithm 1, functions for the shift of a matrix in all the possible directions are required.

The selection of a mate for the cell (x, y) happens in lines 8-10 of Algorithm 1, where the cell with best fitness value and lowest order number among those in the neighborhood is chosen. Its state and fitness values are saved as *selp* and *self*, respectively.

Crossover. Interactions between cells (considering states and fitness values) are described through an operator `co` that is the implementation of the crossover operator descibed in Eq. (3) (line 11 in Algorithm 1).

Mutation. The mutation operator depends on two variables: *fmut* and *pmut* which give respectively its frequency (how many steps pass between a mutation and another) and the probability of its occurring. If mutation is possible and a randomly generated number is less than *pmut*, the state of the cell is changed with one of the possible ones (lines 12-13 in Algorithm 1).

Algorithm 1: Rules

```
1  order = [0, ..., 8];
2  for t in {1, ..., N} do                                  // start of the run
3      fit = fitfun(pop);                                   // fitness function
4      order = Shuffle(order, seed);
5      pp[order] = [pop, N(pop), N(E(pop)), E(pop), S(E(pop)),
         S(pop), S(W(pop)), W(pop), N(W(pop))];
6      ff[order] = [fit, N(fit), N(E(fit)), E(fit), S(E(fit)), S(fit),
         S(W(fit)), W(fit), N(W(fit))];
7      for x in {1, ..., m} and y in {1, ..., n} do
           // selection
8          best = Min(i | ff[i, x, y] = Max(ff[order, x, y]));
9          selp = pp[best[x, y], x, y];
10         self = ff[best[x, y], x, y];
           // crossover
11         pop[x, y] = co(pop[x, y], fit[x, y], selp, self, maxp, maxf);
12         if 0 ≤ Random(0, 1, seed) < pmut and t mod nmut = 0 then
               // mutation
13             pop[x, y] = Int(Random(0, maxp, seed));
```

3. Game of Life implementation and generalizations

As we say in Section 1, the main example of two-dimensional Cellular Automaton is Conway's Game of Life [14] and here we show one of the main results reached with our system: it is possible to implement Conway's rule using cellular evolution. Then, we present two brand new Cellular Automata inspired by Game of Life, describing possible interactions between two lives

in the same environment. For both cases the implementation with cellular evolution is explained.

3.1. Game of Life with cellular evolution

In the 1960s, John H. Conway wanted to define a dynamical rule satisfying the following three criteria:

- it should be difficult to prove that a pattern grows without limit;
- not all simple initial states should immediately yield trivial final states;
- there should exist simple patterns that evolve for many iterations before settling into a simple final state.

After a great deal of experimentation, Conway finally settled on the well known two-dimensional rule of Game of Life.

Game of Life. A two-dimensional Cellular Automaton which evolves in a squared lattice of binary-valued cells and uses Moore neighborhoods. Its dynamical rule is defined as:

- the *birth* of a cell (passage from 0 to 1) occurs if it has exactly three living cells in its neighborhood;
- a cell encounters *death* (passage from 1 to 0) for isolation if it has less than two living cells in its neighborhood, or for overcrowding when they are more than three;
- a living cell *survives* if it is surrounded by two or three living cells.

We can write this rule with the following formalism:

$$\begin{aligned} \sigma_{\underline{i}}(t+1) &= \Phi_{GoL}(\mathcal{C}_{\underline{i}}(t)) = \\ &= \begin{cases} 1 & \text{if } \sum_{\underline{j}\in\mathcal{N}(\underline{i})} \sigma_{\underline{j}}(t) = 3, \\ \sigma_{\underline{i}}(t) & \text{if } \sum_{\underline{j}\in\mathcal{N}(\underline{i})} \sigma_{\underline{j}}(t) = 2, \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{5}$$

What distinguishes this system from all the others and makes this rule truly remarkable is that GoL has been proven to be capable of *universal computation* [35]. The principal consequence of this statement is that Game of

Life can carry out arbitrary algorithmic procedures, so that it can be used in place of every standard digital computer. Furthermore, this property gives us an important information about Game of Life's dynamical complexity: this rule is actually capable of displaying arbitrarily complicated behavior and generally there is no short-cut route to the final outcome of its evolution.

Now that we are more familiar with Conway's rule it is possible to talk about its implementation using cellular evolution. Our system can be customized changing its two operators: the fitness function and the crossover matrix. Here we want to show one of the possible choices to obtain Game of Life in cellular evolution.

In Game of Life two states are consider, so that $\Sigma = \{0, 1\}$. We observe that the future state of a cell depends only by the state of its neighbors. Hence we choose to use a fitness function with values in $\mathcal{F} = \{0, 1\}$, which describes the will of each cell: if its fitness value is 0 then the cell wants to die or stay dead while if it is 1 the cell will become or stay alive. The simplicity of Game of Life rule leads to a trivialization of the crossover operator: given a fitness value, the result consists simply in making the wish of the cell comes true, without looking at the state or the fitness value of the mate chosen by the cell. In practice, the operators for implementing Game of Life in cellular evolution are:

- $f : \Sigma^9 \longrightarrow \mathcal{F}$ given by

$$f_{\underline{i}}(t) = \Phi_{GoL}(\mathcal{C}_{\underline{i}}(t)) = \begin{cases} 1 & \text{if } \sum_{\underline{j} \in \mathcal{N}(\underline{i})} \sigma_{\underline{j}}(t) = 3, \\ \sigma_{\underline{i}}(t) & \text{if } \sum_{\underline{j} \in \mathcal{N}(\underline{i})} \sigma_{\underline{j}}(t) = 2, \\ 0 & \text{otherwise;} \end{cases} \tag{6}$$

- $\chi : \Sigma \times \mathcal{F} \times \Sigma \times \mathcal{F} \times \mathbb{N} \times \mathbb{N} \longrightarrow \Sigma$ where:

$$\chi(\sigma_{\underline{i}}(t), f_{\underline{i}}(t), \sigma_{\underline{j}}(t), f_{\underline{j}}(t), 1, 1) = f_{\underline{i}}(t), \tag{7}$$

  or, giving all the possible cases, we have Table 1.

In conclusion, we were able to obtain Conway's rule in our system, which means that cellular evolution is capable of universal computation.

χ		$(\sigma_{\underline{i}}(t), f_{\underline{i}}(t))$			
		(0,0)	(0,1)	(1,0)	(1,1)
$(\sigma_{\underline{j}}(t), f_{\underline{j}}(t))$	(0,0)	0	1	0	1
	(0,1)	0	1	0	1
	(1,0)	0	1	0	1
	(1,1)	0	1	0	1

Table 1: Crossover operator to implement Game of Life and its new 3-states version with cellular evolution.

3.2. Two variants of Game of Life

Here we introduce two new variants of Conway's Game of Life. In particular, these new rules maintain the bi-dimensional lattice but consider more than two possible states, while the dynamical rules are based on the original one. These systems are inspired by natural interactions between two species. As before, we implement these new Cellular Automata with cellular evolution, in order to show its potentiality.

3.2.1. Three states evolution

In this case there are three possible states: $\Sigma = \{0, 1, 2\}$, where 0 means dead while 1 and 2 are the two possible living states.

The evolution is described by the following rule:

- a dead cell ($\sigma_{\underline{i}}(t) = 0$) changes state if there are exactly 3 cells alive in its neighborhood. In that case its state value becomes 1 if in its neighborhood there are more cells with state value 1 than those with state values 2, and 2 otherwise;
- a living cell ($\sigma_{\underline{i}}(t) > 0$) stay unchanged if there are 2 or 3 cells alive in its neighborhood, otherwise it dies.

We can summarize the evolution of the system considering for each cell $\underline{i}$ the values $\mathbf{1}_{\underline{i}}(t) = |\{1 \in \mathcal{C}_{\underline{i}}(t)\}|$ and $\mathbf{2}_{\underline{i}}(t) = |\{2 \in \mathcal{C}_{\underline{i}}(t)\}|$. Hence the dynamic rule becomes:

$$\sigma_{\underline{i}}(t+1) = \Phi_3(\mathcal{C}_{\underline{i}}(t)) = \\ = \begin{cases} 1 & \text{if } \sigma_{\underline{i}}(t) = 0,\ \mathbf{1}_{\underline{i}}(t) + \mathbf{2}_{\underline{i}}(t) = 3 \text{ and } \mathbf{1}_{\underline{i}}(t) > \mathbf{2}_{\underline{i}}(t), \\ 2 & \text{if } \sigma_{\underline{i}}(t) = 0,\ \mathbf{1}_{\underline{i}}(t) + \mathbf{2}_{\underline{i}}(t) = 3 \text{ and } \mathbf{1}_{\underline{i}}(t) < \mathbf{2}_{\underline{i}}(t), \\ \sigma_{\underline{i}}(t) & \text{if } \sigma_{\underline{i}}(t) > 0 \text{ and } 1 < \mathbf{1}_{\underline{i}}(t) + \mathbf{2}_{\underline{i}}(t) < 4, \\ 0 & \text{otherwise.} \end{cases} \quad (8)$$

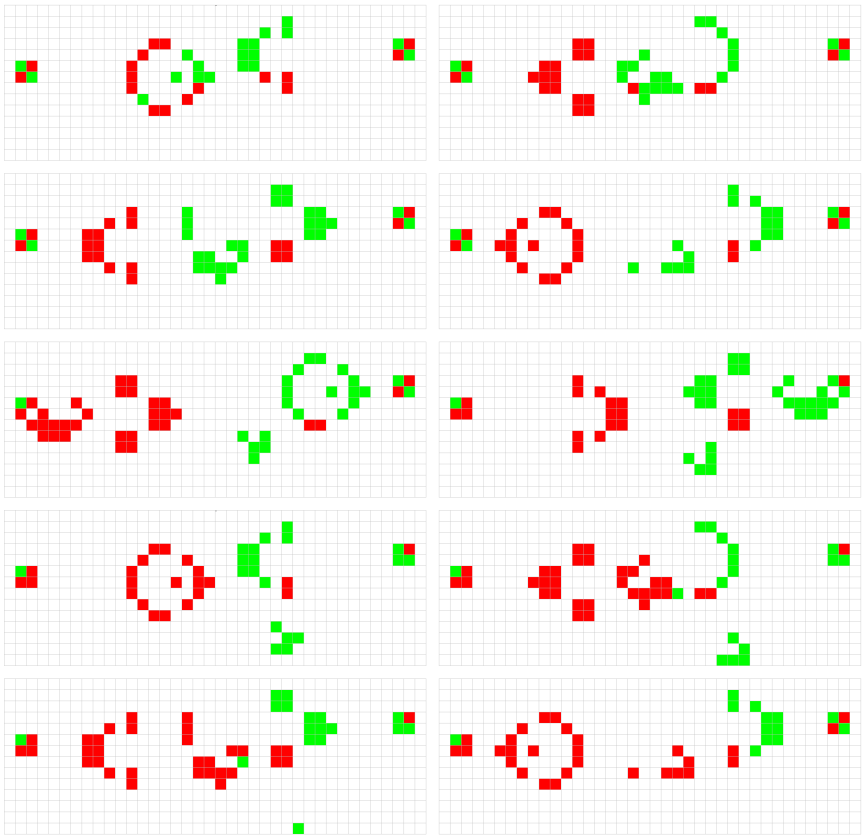


Figure 2: A version of the glider gun in three colors Game of Life seen each 5 time-steps.

For the implementation using cellular evolution we can use operators similar to those used for the original Game of Life:

- $f : \Sigma^9 \longrightarrow \mathcal{F}$ given by $f_{\underline{i}}(t) = \Phi_3(\mathcal{C}_{\underline{i}}(t))$;
- $\chi : \Sigma \times \mathcal{F} \times \Sigma \times \mathcal{F} \times \mathbb{N} \times \mathbb{N} \longrightarrow \Sigma$ where

$$\chi(\sigma_{\underline{i}}(t), f_{\underline{i}}(t), \sigma_{\underline{j}}(t), f_{\underline{j}}(t), 2, 2) = f_{\underline{i}}(t), \tag{9}$$

whose possible cases are described in Table 1.

All the studies on the classic Game of Life can be repeated on this three-states version. For example, a glider gun can be obtained with the pattern in Fig. 2: it spawns one green glider followed by infinite red ones.

3.2.2. Four states evolution

This case differs from the previous one since there are four possible state values: $\Sigma = \{0, 1, 2, 3\}$, where 0 and 1 are dead and alive state for the first form of life while 2 and 3 are dead and alive state for the second one. In this case the two form of life evolve and live in symbiosis, that is they help each other to stay alive.

The rule describing the behavior of this system is given by:

- a dead cell of the first form of life ($\sigma_{\underline{i}}(t) = 0$) will become alive (with state value 1) if in its neighborhood there are exactly 3 cells with state value 1 or if there are exactly 5 cells with states 0 or 2;
- for a dead cell of the second form of life ($\sigma_{\underline{i}}(t) = 2$) the behavior is analogous: it will become alive ($\sigma_{\underline{i}}(t+1) = 3$) if in its neighborhood there are exactly 3 cells with state value 3 or or if there are exactly 5 cells with states 0 or 2;
- a living cell of the first form of life ($\sigma_{\underline{i}}(t) = 1$) stays alive if in its neighborhood there are 2 or 3 cells with its state value or if there is a quantity between 1 and 4 of cells with state value 3, otherwise it dies;
- the behavior of a living cell of the second form of life ($\sigma_{\underline{i}}(t) = 3$) is similar: it stays alive if, in its neighborhood, there are 2 or 3 cells with state value 3 or if there is a quantity between 1 and 4 of cells with state value 1, otherwise it dies.

In order to implement this system, for each cell $\underline{i}$ two values are considered: $\mathbf{1}_{\underline{i}}(t) = |\{1 \in \mathcal{C}_{\underline{i}}(t)\}|$ and $\mathbf{3}_{\underline{i}}(t) = |\{3 \in \mathcal{C}_{\underline{i}}(t)\}|$. They allow us to write the dynamic rule as follows:

$$\sigma_{\underline{i}}(t+1) = \Phi_4(\mathcal{C}_{\underline{i}}(t)) = \\ = \begin{cases} 0 & \text{if } \sigma_{\underline{i}}(t) = 1,\ \mathbf{1}_{\underline{i}}(t) \notin \{2,3\} \text{ and } \mathbf{3}_{\underline{i}}(t) \notin \{1,2,3,4\}, \\ 1 & \text{if } \sigma_{\underline{i}}(t) = 0 \text{ and } \mathbf{1}_{\underline{i}}(t) = 3 \text{ or } \mathbf{1}_{\underline{i}}(t) + \mathbf{3}_{\underline{i}}(t) = 3, \\ 2 & \text{if } \sigma_{\underline{i}}(t) = 3,\ \mathbf{3}_{\underline{i}}(t) \notin \{2,3\} \text{ and } \mathbf{1}_{\underline{i}}(t) \notin \{1,2,3,4\}, \\ 3 & \text{if } \sigma_{\underline{i}}(t) = 2 \text{ and } \mathbf{3}_{\underline{i}}(t) = 3 \text{ or } \mathbf{1}_{\underline{i}}(t) + \mathbf{3}_{\underline{i}}(t) = 3, \\ \sigma_{\underline{i}}(t) & \text{otherwise.} \end{cases} \tag{10}$$

Now we implement this system with cellular evolution. We may use operators analogous to those used until now, but instead we choose to change

χ		$(\sigma_{\underline{i}}(t), f_{\underline{i}}(t))$							
		(0,0)	(0,1)	(1,0)	(1,1)	(2,0)	(2,1)	(3,0)	(3,1)
$(\sigma_{\underline{j}}(t), f_{\underline{j}}(t))$	(0,0)	0	1	0	1	2	3	2	3
	(0,1)	0	1	0	1	2	3	2	3
	(1,0)	0	1	0	1	2	3	2	3
	(1,1)	0	1	0	1	2	3	2	3
	(2,0)	0	1	0	1	2	3	2	3
	(2,1)	0	1	0	1	2	3	2	3
	(3,0)	0	1	0	1	2	3	2	3
	(3,1)	0	1	0	1	2	3	2	3

Table 2: Crossover operator to implement the 4-states version of Game of Life with cellular evolution.

a bit both of them. As we did in the original Game of Life case we still use a fitness function that describes the will of a cell to become or stay alive but its possible values are only two: $\mathcal{F} = \{0, 1\}$. While the fitness value gives the will, the crossover matrix decides, depending on the current state of the cell, what is its form of life and consequently what will be its new state value. In practice, we have:

- $f : \Sigma^9 \longrightarrow \mathcal{F}$ given by

$$f_{\underline{i}}(t) = \Phi_4(\mathcal{C}_{\underline{i}}(t)) (\text{mod } 2); \tag{11}$$

- $\chi : \Sigma \times \mathcal{F} \times \Sigma \times \mathcal{F} \times \mathbb{N} \times \mathbb{N} \longrightarrow \Sigma$ where

$$\chi(\sigma_{\underline{i}}(t), f_{\underline{i}}(t), \sigma_{\underline{j}}(t), f_{\underline{j}}(t), 3, 1) = \begin{cases} f_{\underline{i}}(t) & \text{if } \sigma_{\underline{i}}(t) \in \{0, 1\}, \\ f_{\underline{i}}(t) + 2 & \text{if } \sigma_{\underline{i}}(t) \in \{2, 3\}. \end{cases} \tag{12}$$

  Every possible case is collected in Table 2.

As for the three-states version, all the studies on the classic Game of Life can be repeated on this new Cellular Automaton. For example, a life generator can be obtained with the pattern in Fig. 3: the simple red structure at step 0 creates infinite patterns of green cells (like the R-pentomino in the original Game of Life).

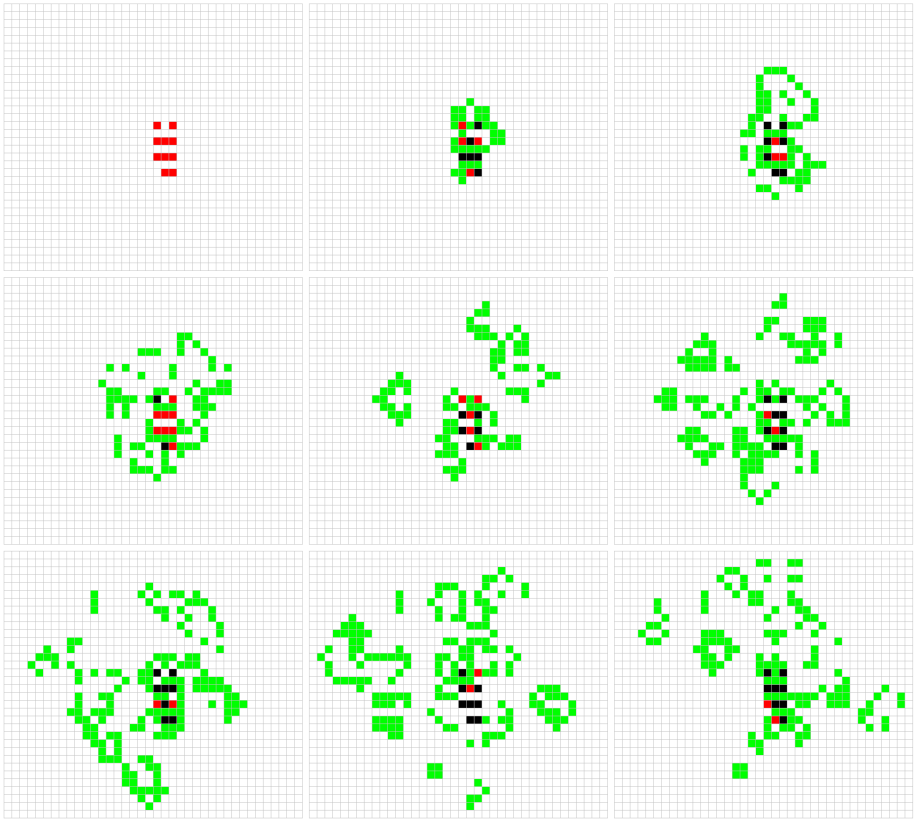


Figure 3: Steps 0, 10, ... , 80 of a life generator in the 4-states variant of Game of Life

4. Prisoner's dilemma

In this section we study the prisoner's dilemma game by means of our cellular automaton. This game is largely treated in the literature in several ways and widely used as an interesting example of evolutionary game, see, e.g., [2], [20]. The basic idea is that two prisoners are under questioning in two separate rooms. They can or betray the partner by confessing the truth, or cooperate by staying silent. Depending on their behaviors, they can be sentenced to different amount of years in prison. In particular, the best pay-off is get by defecting a cooperative prisoner (dc), followed by the one obtained if they cooperate (cc), then by the case in which they are both defective (dd) and finally by the pay-off get by a cooperative prisoner that has been betrayed (cd). As treated in [30, 41], this game can be generalized in a multidimensional lattice by considering as gain the total pay-off obtained by playing with the adjacent neighbors. The evolution of the population is characterized by the rule according to which, among all the strategies of the neighbors and the current strategy, the one with highest gain is adopted for the following generation.

This system can be easily implemented in cellular evolution, with:

- the local value space $\Sigma = \{c = 0, d = 1\}$;
- $f_{dc,cc,dd,cd} : \Sigma^9 \longrightarrow \mathcal{F}$ exploits $C = |\{c \in \mathcal{C}_{\underline{i}}(t)\}|$ and $D = |\{d \in \mathcal{C}_{\underline{i}}(t)\}|$ and is given by

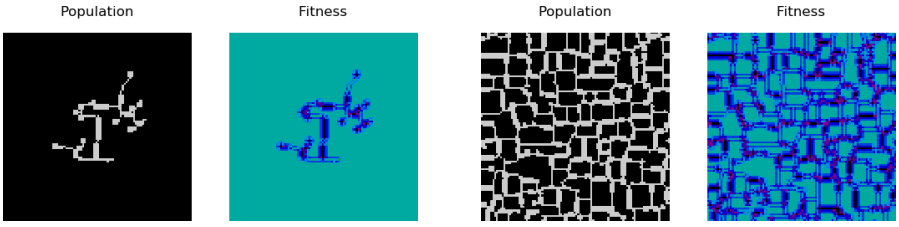
$$f_{\underline{i}}(t) = \begin{cases} dc \cdot C + dd \cdot (D-1) & \text{if } \sigma_{\underline{i}}(t) = d\,, \\ cc \cdot (C-1) + cd \cdot D & \text{if } \sigma_{\underline{i}}(t) = d\,; \end{cases} \tag{13}$$

- $\chi : \Sigma \times \mathcal{F} \times \Sigma \times \mathcal{F} \times \mathbb{N} \times \mathbb{N} \longrightarrow \Sigma$ where

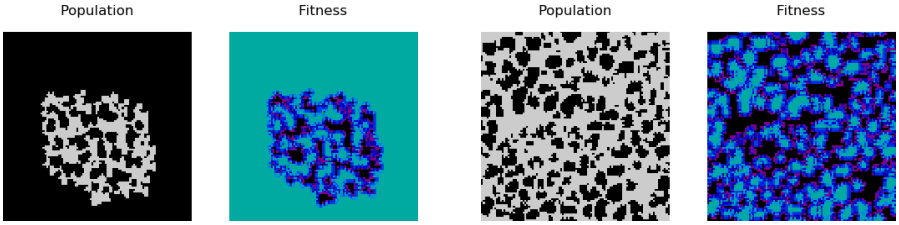
$$\chi(\sigma_{\underline{i}}(t), f_{\underline{i}}(t), \sigma_{\underline{j}}(t), f_{\underline{j}}(t), 1, 8 \cdot dc) = \sigma_{\underline{j}}(t)\,. \tag{14}$$

Among the possible choices for the parameters, one of the most studied considers $cd = dd = 0, cc = 1$ and has degree of freedom $dc > 1$. In this case, interesting results can be obtained by studying the evolution of a single 2×2 square with state d in a population of c's in a toroidal world of size 100×100. By changing dc, the system converges to:

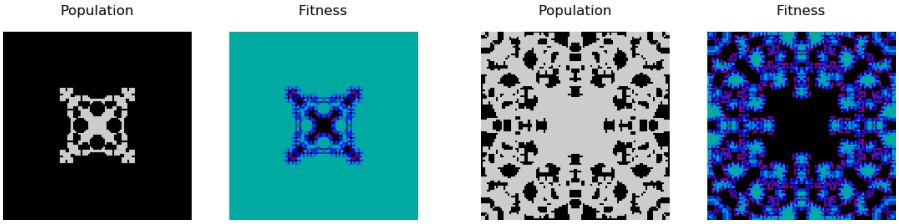
- a full world of c's when $dc < 1.4$;



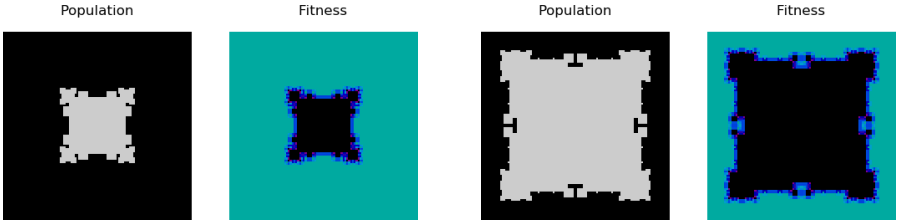
(a) $dc = 1.4$, generation 200 and 1000.



(b) $dc = 1.6$, generation 50 and 1000.

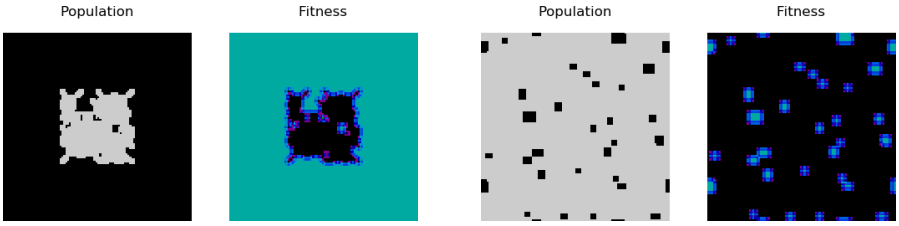


(c) $dc = 1.65$, generation 20 and 1000.

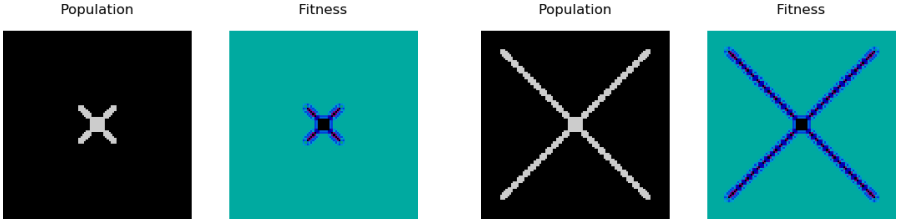


(d) $dc = 1.7$, generation 20 and 40.

Figure 4: Behaviors for the classic bi-dimensional prisoner's dilemma starting from an initial 2×2 square of d's (grey) in a population of c's (black).



(a) $dc = 1.75$, generation 20 and 60.



(b) $dc = 1.8$, generation 10 and 40.

Figure 5: Other behaviors for the classic bi-dimensional prisoner's dilemma starting from an initial 2×2 square of d's (grey) in a c's (black) world.

- a constantly changing interlaced network with a structurally static pattern for $dc = 1.4$ (Fig. 4a). This is due to the random selection among fittest cells, since the fitness of the initial d's is equal to that of the c's at the four corner of the square ($fit[x, y] = 7$);
- the initial state, which remains fixed, if $1.4 < dc < 1.6$;
- a behavior forever changing from one generation to the following one for $dc = 1.6$ (Fig. 4b). Since it is possible that a d has the same fitness value of a c ($fit[x, y] = 8$), the behavior is very chaotic;
- an ever changing kaleidoscopic behavior for $1.6 < dc < \frac{5}{3}$ (Fig. 4c);
- an expansion of the central area of d's with fractal-like borders for $\frac{5}{3} \leq dc < 1.75$ (Fig. 4d). The expansion should continue forever, but the collisions at the border of the world stop it in finitely many steps;
- an expansion of d's with a jagged square border that leaves behind some static rectangular isles of c's, for $dc = 1.75$ (Fig. 5a);
- an expanding X mark of d's for $1.75 < dc < 2$ (Fig. 5b);

- when $dc = 2$, a behavior similar to that obtained when $dc = 1.75$;
- when $2 < dc < \frac{8}{3}$, a behavior similar to that for $\frac{5}{3} \leq dc < 1.75$;
- a full world of d's for $dc \geq \frac{8}{3}$ through a perfectly squared expansion.

These results are similar to those described in [30], but the random selection (in all the figures, $seed = 3$) creates some different behaviors.

For a 2×2 square of c's in a population of d's, the possible behaviors are only two: total annihilation when $dc \geq 1.5$ and total conquest for $dc < 1.5$.

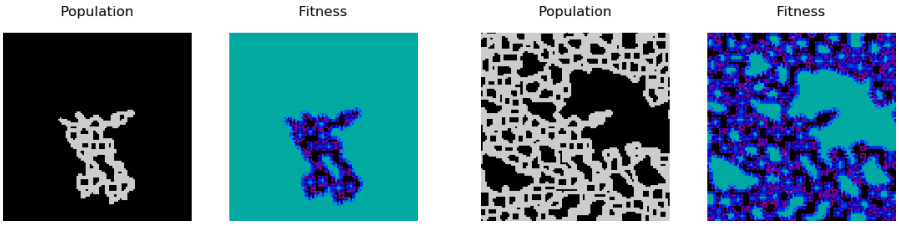
Since the crossover in cellular evolution depends on both states and fitness values, it is possible to obtain new behaviors while maintaining the same fitness function. For example, it is possible to consider a crossover operator that returns as new state the ceil-rounding of the fitness weighed mean between the old state and that of the selected cell, i.e.,

$$\chi(\sigma_{\underline{i}}(t), f_{\underline{i}}(t), \sigma_{\underline{j}}(t), f_{\underline{j}}(t), 1, 8 \cdot dc) = \left\lceil \frac{\sigma_{\underline{i}}(t) \cdot f_{\underline{i}}(t) + \sigma_{\underline{j}}(t) \cdot f_{\underline{j}}(t)}{f_{\underline{i}}(t) + f_{\underline{j}}(t)} \right\rceil . \quad (15)$$

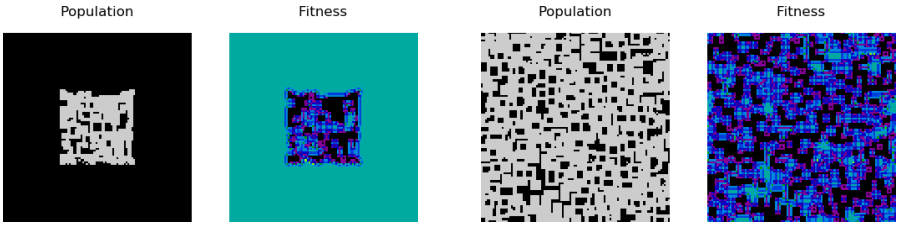
The behaviors obtained from a 2×2 square of d's in a population of c's are different from above:

- for $dc < 1.6$ the initial state is fixed;
- when $dc = 1.6$ there is a completely new chaotic behavior of expansion, which brings to blinking isles of c's in a network of d's (Fig. 6a);
- an expanding X mark arises when $1.6 < dc < 2$, with a rectangular center of c's that can be fixed of blinking;
- for $dc = 2$ there is an expansion with jagged square border that leaves a flashing world behind (Fig. 6b);
- when $2 < dc < \frac{8}{3}$ there is a fractal border expansion that generates a flashing kaleidoscope in its center area (Fig. 6c);
- finally, for all $dc \geq \frac{8}{3}$ there is an expanding square which contains other flashing squares inside (Fig. 6d).

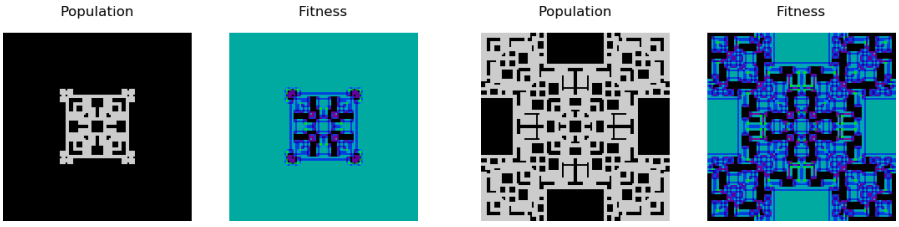
In this case a 2×2 square of c's in a d's world passes immediately to a c's population with at the center a 6×6 square of d's holed with a 2×2 square of c's.



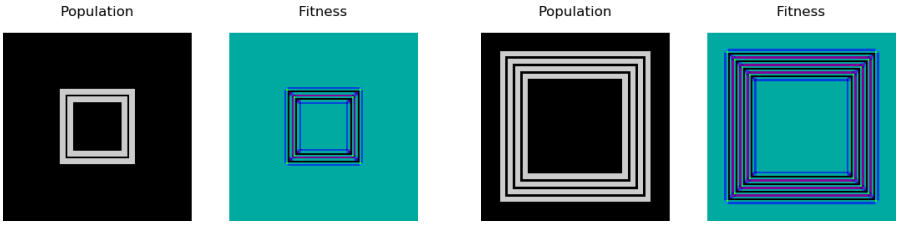
(a) $dc = 1.6$, generation 50 and 1000.



(b) $dc = 2$, generation 20 and 60.



(c) $dc = 2.5$, generation 20 and 60.



(d) $dc = 2.7$, generation 20 and 40.

Figure 6: Behaviors for the bi-dimensional prisoner's dilemma with ceil-rounded fitness-weighed mean as crossover, starting from an initial 2×2 square of d's (grey) in a population of c's (black).

5. Rock-paper-scissors

The two-dimensional system with total gain as fitness function can be based on all kind of 1-on-1 games, and they can all be implemented with cellular evolution. In this section we focus on another classic game: rock-paper-scissors (RPS). This game has been widely studied and used for simulating, e.g., different colonies of bacteria that coexist and interact [11], for studying decision-making in non-cooperative strategic interactions [46] or for representing a food chain in ecosystems [38]. Further studies can be found in [22], [26], [42]. Here, each cell in the population can assume one of three possible states. The fitness function is characterized by three parameters representing the possible gain for each win (ties and losses give gain 0 in order to avoid negative fitness values): *sp* is how much a cell that plays scissors (state 0) wins against a paper (state 1), *pr* is the gain resulting from an encounter between paper and rock (state 2), and finally *rs* represents the gain of rock against scissors. The same crossover operator introduced before is adopted.

Thus, the implementation in cellular evolution has:

- the local value space $\Sigma = \{0, 1, 2\}$;
- $f_{sp,pr,rs} : \Sigma^9 \longrightarrow \mathcal{F}$ exploits $S = |\{0 \in \mathcal{C}_{\underline{i}}(t)\}|$, $P = |\{1 \in \mathcal{C}_{\underline{i}}(t)\}|$ and $R = |\{2 \in \mathcal{C}_{\underline{i}}(t)\}|$ and is given by

$$f_{\underline{i}}(t) = \begin{cases} sp \cdot P & \text{if } \sigma_{\underline{i}}(t) = 0\,, \\ pr \cdot R & \text{if } \sigma_{\underline{i}}(t) = 1\,, \\ rs \cdot S & \text{if } \sigma_{\underline{i}}(t) = 2\,; \end{cases} \tag{16}$$

- $\chi : \Sigma \times \mathcal{F} \times \Sigma \times \mathcal{F} \times \mathbb{N} \times \mathbb{N} \longrightarrow \Sigma$ where

$$\chi\big(\sigma_{\underline{i}}(t), f_{\underline{i}}(t), \sigma_{\underline{j}}(t), f_{\underline{j}}(t), 2, 8 \cdot \max(sp, pr, rs)\big) = \sigma_{\underline{j}}(t)\,. \tag{17}$$

When all the possible strategies have the same gain, e.g. $sp = pr = rs = 1$, the generic behavior consists in an eternal fight among the strategies with spiral shape areas (Fig. 7).

Interesting patterns and behaviors arise when considering different gain parameters. In this case, we can think to the RPS game as a model "prey–predator" with three different populations, where the gain of a predator can be higher that the gain of another predator. The following cases have

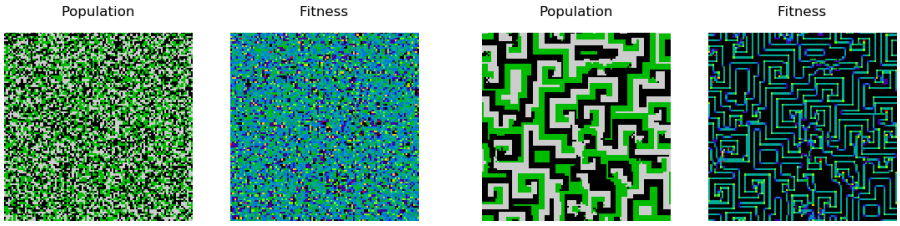


Figure 7: RPS evolution from a random 100×100 population ($seed = 3$) to step 1000. The possible states are scissors (0) in black, paper (1) in green and rock (2) in grey.

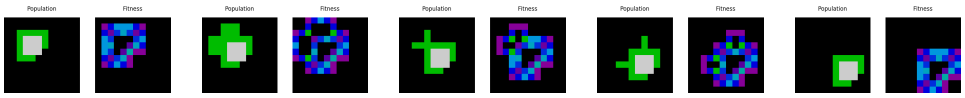


Figure 8: First 5 generations of a glider in RPS with $sp = 1, pr = 1.1$ and $rs = 1.2$.

$sp \leq pr \leq rs$, but they can be generalized by permuting the gain values and the states in the population.

In all the instances it is possible to obtain a glider. One of the simplest gliders arise with $sp = 1, pr = 1.1, rs = 1.2$. It is shown in Fig. 8 and has period 4.

When starting with the same initial structure but with a different instance of RPS with $sp < pr < rs$, the evolution can generate a pseudo-glider (Fig. 9), characterized by a changing shape (due to the random selection among fittest cells) that always contains the initial one, or to new gliding shapes with different periods (Fig. 10).

With some choices of parameters there are configurations that work as life generators. For example, when $sp = 1.7, pr = 3, rs = 5$, the initial state in Fig. 11 generates an infinitely expanding pattern with lots of gliders.

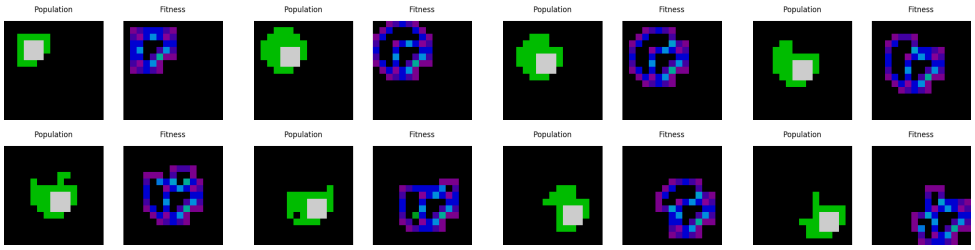


Figure 9: The first 8 steps of a pseudo-glider in RPS with $sp = 1, pr = 1.5$ and $rs = 2$, generated from the initial configuration used in Fig. 8.

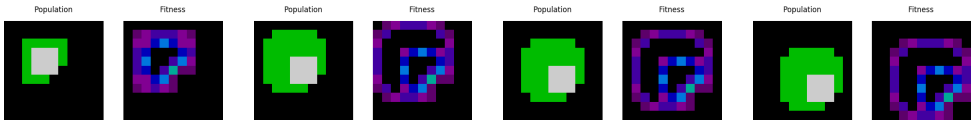


Figure 10: The rise of a glider with period 1 in RPS with $sp = 1, pr = 2$ and $rs = 3$, from the initial configuration used in Fig. 8.

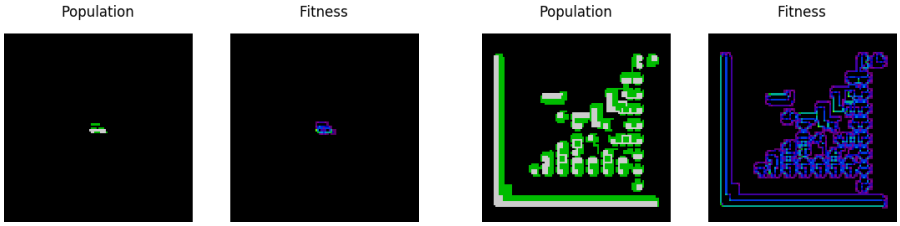


Figure 11: The evolution of a life generator in RPS with $sp = 1.7, pr = 3$ and $rs = 5$, from the initial pattern to step 40.

As for the ceil-rounded fitness-weighed mean introduced as crossover in the bi-dimensional prisoner's dilemma, new crossover operators depending on both state and fitness values can be adopted with RPS. In this case, interesting behaviors are obtained when considering a function that returns as new state the fitness-weighed mean between the old state and that of the selected cell, rounded down if the sum of the states is equal or below the maximum state and rounded up elsewhere. This crossover operator is $\chi\big(\sigma_{\underline{i}}(t), f_{\underline{i}}(t), \sigma_{\underline{j}}(t), f_{\underline{j}}(t), 2, 8 \cdot \max(sp, pr, rs)\big) = \sigma_{\underline{i}}(t+1)$ where

$$\sigma_{\underline{i}}(t+1) = \begin{cases} \left\lfloor \frac{\sigma_{\underline{i}}(t) \cdot f_{\underline{i}}(t) + \sigma_{\underline{j}}(t) \cdot f_{\underline{j}}(t)}{f_{\underline{i}}(t) + f_{\underline{j}}(t)} \right\rfloor & \text{if } \sigma_{\underline{i}}(t) + \sigma_{\underline{j}}(t) \leq 2\,, \\ \left\lceil \frac{\sigma_{\underline{i}}(t) \cdot f_{\underline{i}}(t) + \sigma_{\underline{j}}(t) \cdot f_{\underline{j}}(t)}{f_{\underline{i}}(t) + f_{\underline{j}}(t)} \right\rceil & \text{if } \sigma_{\underline{i}}(t) + \sigma_{\underline{j}}(t) > 2\,. \end{cases} \tag{18}$$

With this operator, the simple pattern that behaved as a glider in the previous examples generates an entire expansion of fighting cells, as depicted in Fig. 12.

6. Conclusions

In this work we introduce and give some properties of cellular evolution, a new kind of cellular automaton inspired by genetic algorithms, which appears to be powerful and flexible, allowing to easily implement and study many different systems.

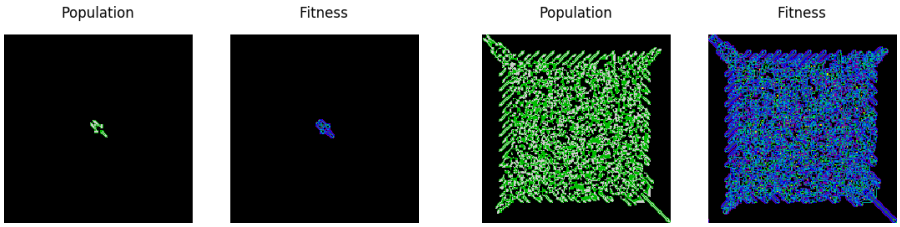


Figure 12: Expansion of the glider pattern in RPS with $sp = 1, pr = 1.1$ and $rs = 1.2$ and crossover operator given in Eq. (18), generation 10 and 100.

In particular we concentrate on its definition and some useful implementations, like the well known Game of Life created by Conway that gives us the property of universal computation. Moreover, we propose the implementation of two variants of the Game of Life: one with three possible states, where a glider gun is shown, and the other with four possible states, with which a life generator is described. Then, we focus on two classical games widely studied in the literature also for testing cellular automata properties. Firstly, we study the prisoner's dilemma in a two-dimensional lattice. We show some known and new behaviors by changing the parameters and also by introducing a new crossover operator that specifically exploits the features of cellular evolution. Secondly, the classic game of rock-paper-scissors is considered: here we focus on studying the general behavior and we find some gliders depending on the chosen parameters. In addition, another new crossover operator allows us to find other interesting behaviors. At https://github.com/duttos/cellular_evolution, the reader can also find videos that describe the cases studied in Sections 3, 4 and 5.

These are only some of the systems that can be implemented with our system, but the flexibility of cellular evolution can be exploited to implement many other case studies in order to further deepen the knowledge about this powerful system. In future work, it would be interesting to deepen the study of lattice games considering iterated games (with more than one step) and more sophisticated strategies (based on past movies). Moreover, another interesting aspect to be explored regards the exploration of probabilistic rules.

Acknowledgment

We would like to thank the anonymous referees for the valuable suggestions and corrections that allowed to improve the paper.

References

[1] Alonso-Sanz, R., A structurally dynamic cellular automaton with memory, Chaos, Solitons and Fractals, vol. 32, 1285–1295, 2007.

[2] R. Axelrod, Effective choice in the prisoner's dilemma, Journal of Conflict Resolution, vol. 24, no. 1, 3–25, 1980.

[3] T. Back, R. Breukelaar, Using genetic algorithms to evolve behavior in cellular automata, Lecture Notes in Computer Science, vol. 3699, 1–10, 2005.

[4] V. Barmpoutis, A compact self-organizing cellular automata-based genetic algorithm, arXiv:0711.2478v1, 2007.

[5] Benjamin, S. C., Johnson, N. F., Hui, P. M., Cellular automata models of traffic flow along a highway containing a junction, Journal of Physics A: Mathematical and General, vol. 29, 3119–3127, 1996.

[6] Beros, A., Monique, C., Kari, N., Co-evolving cellular automata for morphogenesis, Discrete Contin. Dyn. Syst. Ser. B, vol. 24, no. 5, 2053–2071, 2019.

[7] Bosch, R. A., Maximum density stable patterns in variants of Conway's game of Life, Operations Research Letters, vol. 27, 7–11, 2000.

[8] Chaudhary, B., Singh, K., Pseudonym generation using genetic algorithm in vehicular ad hoc networks, J. Discrete Math. Sci. Cryptogr., vol. 22, no. 4, 661–677, 2019.

[9] Coronel, A., Berres, S., Lagos, R., Calibration of a sedimentation model through a continuous genetic algorithm, Inverse Probl. Sci. Eng., vol. 27, no. 9, 1263–1278, 2019.

[10] Daoudia, A. K., Moussa, N., Numerical simulations of a three-lane trafficmodel using cellular automata, Chinese Journal of Physics, vol. 41, no. 6, 2003.

[11] P. G. Esteban, A. Rodriguez-Paton, Simulating a rock-scissors-paper bacterial game with a discrete cellular automaton, Lecture Notes in Computer Science, vol. 6687, 363–370, 2011

[12] M. Giacobini, M. Tomassini, A. G. B. Tettamanzi, E. Alba, Selection intensity in cellular evolutionary algorithms for regular lattices, IEEE Transactions on evolutionary comptation, vol. 9, no. 5, 2005.

[13] Gotts, N. M., Ramifying Feedback Networks, Cross-Scale Interactions, and Emergent Quasi Individuals in Conway's Game of Life, Artificial Life, vol. 15, no. 3, 2009.

[14] Gardner, M., Mathematical Games – The fantastic combinations of John Conway's new solitaire game "life", Scientific American, vol. 223, no. 4, 1970.

[15] Gomez-Ramirez, E., Pazienza, G. E., The Game of Life Using Polynomial Discrete Time Cellular Neural Networks, Analysis and Design of Intelligent Systems using Soft Computing Techniques, vol 41. Springer, Berlin, Heidelberg, 719–726, 2007.

[16] Holland, J., Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.

[17] Ibarra, O. H., Palis, M. A., Kim, S. M., Fast Parallel Language Recognition by Cellular Automata, Theoretical Computer Science, vol. 41, 231–246, 1985.

[18] Kamalika, B., Sukanta, D., Random number generation using decimal cellular automata, Commun. Nonlinear Sci. Numer. Simul., vol. 78, 2019.

[19] Krawczyk, M. J., New aspects of symmetry of elementary cellular automata, Chaos, Solitons and Fractals, vol. 78, 86–94, 2015.

[20] S. Le, R. Boyd, Evolutionary dynamics of the continous iterated prisoner's dilemma, Journal of Theoretical Biology, vol. 245, 258–267, 2007.

[21] K. Lindgren, Evolutionary dynamics in game–theoretic models, W.B. Arthur, S. Durlauf, D.A. Lane (Eds.), The Economy as an Evolving Complex System II, Santa Fe Institute, Perseus Books, Reading, MA, 1997.

[22] B. H. de Menibus, Y. Le Borgne, Asymptotic behaviour of the one-dimensional rock-paper-scissors cyclic cellular automaton, Available at https://hal.archives-ouvertes.fr/hal-02084842/document, 2019.

[23] Mirzaee, H., Linear combination rule in genetic algorithm for optimization of finite impulse response neural network to predict natural chaotic time series, Chaos, Solitons and Fractals, vol. 41, no. 5, 2681–2689, 2009.

[24] M. Mitchell, J. P. Crutchfield, P. T. Hraber, Evolving cellular automata to perform computations: mechanisms and impediments, Physica D: Nonlinear Phenomena, vol. 75, no. 1–3, 361–391, 1994.

[25] M. Mitchell, J. P. Crutchfield, 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 (EvCA'96) , 1996.

[26] , T. Nagatani, G. Ichinose, K. Tanaka, Heterogeneous network promotes species coexistence: metapopulation model for rock-paper-scissors game, Sci.Rep., vol. 8, 2018.

[27] Nagel, K., Wolf, D. E., Wagner, P., Simon, P., Two-lane traffic rules for cellular automata: A systematic approach, Physical Review E, vol. 58, no. 2, 1425–1437, 1998.

[28] Nandi, S., Kar, B. K., Chaudhuri, P. P., Theory and applications of cellular automata in cryptography, IEEE Transactions on Computers, vol. 43, no. 12, 1346–1357, 1994.

[29] Von Neumann, J., The general and logical theory of automata, Cerebral Mechanisms in Behavior - The Hixon Symposium, 1951.

[30] M. A. Nowak, R. M. May, Evolutionary games and spatial chaos, Nature, vol. 359, 826–829, 1992.

[31] L. Pagie, M. Mitchell, A comparison of evolutionary and coevolutionary search, Int. J. Computat. Intell. Applic., vol. 2, no. 1, 53–69, 2002.

[32] Pazienza, G. E., Gomez-Ramirez, E., Threshold of complexity in 2D binary Cellular Automata found through Polynomial CNNs, Proceedings of the 12th International Workshop on Cellular Nanoscale Networks and their Applications, Berkeley, CA, USA , 2010.

[33] Raptis, T. E., Spectral representations and global maps of cellular automata dynamics, Chaos, Solitons and Fractals, vol. 91, 503–510, 2016.

[34] Reiter, C. A., Cyclic cellular automata in 3D, Chaos, Solitons and Fractals, vol. 44, 764–768, 2011.

[35] Rendell, Paul W., "A Universal Turing Machine in Conway's Game of Life," in *International Conference on High Performance Computing & Simulation*, 2011.

[36] Selman, U., Ecem, A., Shovkat, R., 2D triangular von Neumann cellular automata with periodic boundary, Internat. J. Bifur. Chaos Appl. Sci. Engrg., vol. 29, no. 3, 2019.

[37] M. Sipper, E. Ruppin, Co-evolving architectures for cellular machines, Physica D: Nonlinear Phenomena, vol. 99, no. 4, 428–441, 1997.

[38] K. Tainaka, Physics and ecology of rock-paper-scissors game, Lecture Notes in Computer Science, vol. 2063, 384–395, 2000.

[39] M. Sipper, M. Tomassini, Co-evolving parallel random number generators, Lecture Notes in Computer Science, vol. 1141, 950–959, 1996.

[40] Smith, A. R., Real-time Language Recognition by One-Dimensional Cellular Automata, J. Comput. Syst. Science, vol. 6, 233–253, 1972.

[41] G. Szabo, C. Toke, Evolutionary prisoner's dilemma game on a square lattice, Physical Review E, vol. 58, no. 1, 69–73, 1998.

[42] L. Varga, J. Vukov, G. Szabo, Self-organizing patterns in an evolutionary rock-paper-scissors game for stochastic synchronized strategy updates, Physical Review E, vol. 90, Article 042920, 2014.

[43] Wei, J., Zhou, H., Meng, J., Zhamg, F., Chen, Y., Zhou, S., The SOC in cells' living expectations of Conway's Game of Life and its extended version, Chaos, Solitons and Fractals, vol. 89, 348–352, 2016.

[44] Wolfram, S., A new kind of science, Wolfram Media Inc., 2002.

[45] Wu, M. S., Teng, W. C., Jeng, J. H., Hsieh, J. G., Spatial correlation genetic algorithm for fractal image compression, Chaos, Solitons and Fractals, vol. 28, no. 2, 497–510, 2006.

[46] H. J. Zhou, The rock–paper–scissors game, Contemporary Physics, vol. 57, no. 2, 151–163, 2016.