

Research of a cellular automaton simulating logic gates by evolutionary algorithms

Emmanuel Sapin , Olivier Bailleux , Jean-Jacques Chabrier

Université de Bourgogne,
LERSIA,
9 avenue A. Savary,
B.P. 47870, 21078 Dijon Cedex, France
{olivier.bailleux, jean-jacques.chabrier}@u-bourgogne.fr
emmanuelsapin@hotmail.com

Abstract. This paper presents a method of using genetic programming to seek new cellular automata that perform computational tasks. Two genetic algorithms are used : the first one discovers a rule supporting gliders and the second one modifies this rule in such a way that some components appear allowing it to simulate logic gates. The results show that the genetic programming is a promising tool for the search of cellular automata with specific behaviors, and thus can prove to be decisive for discovering new automata supporting universal computation.

1. Introduction

Cellular automata are discrete systems in which a population of cells evolves from generation to generation on the basis of local transitions rules. They can simulate simplified forms of life [8][9] or physical systems with discrete time and space and local interactions [5][6][7].

Wolfram showed that one-dimensional cellular automata can present a large spectrum of dynamic behaviours. In “Universality and Complexity in Cellular Automata” [11], he introduces a classification of cellular automata, comparing their behaviour with that of some continuous dynamic systems. He specifies four classes of cellular automata on the basis of qualitative criteria. For all initial configurations, Class 1 automata evolve after a finite time to a homogeneous state where each cell has the same value. Class 2 automata generate simple structures where some stable or periodic forms survive. Class 3 automata’s evolution leads, for most initial states, to chaotic forms. All other automata belong to Class 4. According to Wolfram, automata of Class 4 are good candidates for universal computation.

The only binary automaton currently identified as supporting universal computation is Life, which is in Class 4. Its ability to simulate a Turing machine is proved in [2], using gliders (periodic patterns which, when evolving alone, are reproduced identically after some shift in space), glider guns, and eaters. The glider gun emits a glider stream that carries information and creates logic gates through

collisions. The eaters permit, in absorbing gliders, the creation of logic circuits using any combination of logic gates.

The identification of new automata able to simulate logic circuits is consequently a promising lead in the search for new automata supporting universal computation. In this paper, we show how evolutionary algorithms can be used for the research of automata simulating logic gates and we set out an example.

Section 2 describes in detail the creation of logic gates by Life using gliders, glider guns, and eaters as shown in [2]. The framework of our study is then presented in Section 3. Section 4 describes how evolutionary algorithms can be used for seeking new automata supporting gliders and periodic patterns. Using the proposed approach, we found several rules, such as the one described in Section 5, that support a glider gun. Section 6 describes the use of an evolutionary algorithm for modifying the rule in such a way that it supports an eater. Section 7 presents a discussion about some related works. Finally, in the last section we summarize our results and discuss directions for future research.

2. Simulation of a logic gate

In [2], sufficient components allowing Life to simulate logic gates are laid out. Data streams are encoded by gliders streams (i.e. the absence of gliders represents the value 0 and the presence of gliders represents the value 1). Figure 1 shows the simulation of an AND gate with 2 inputs streams A and B. A pattern called *glider gun*, which emits a new glider every 30 generations, is used by this simulation. The glider gun creates a glider stream that "crashes" stream A. If a glider is present in stream A, the two gliders are destroyed by this collision, else the glider emitted by the gun continues its run. The stream resulting from this first collision, at a right angle to stream A, is not(A). This stream crashes stream B producing two streams:

- The first, which is aligned with the stream B, is the result of the operation "A and B".
- The second one is destroyed by a pattern called eater. When a glider crash a eater, the eater survives and the glider dies.

The synchronization and the position of the different components are critical to the proper function of the simulation.

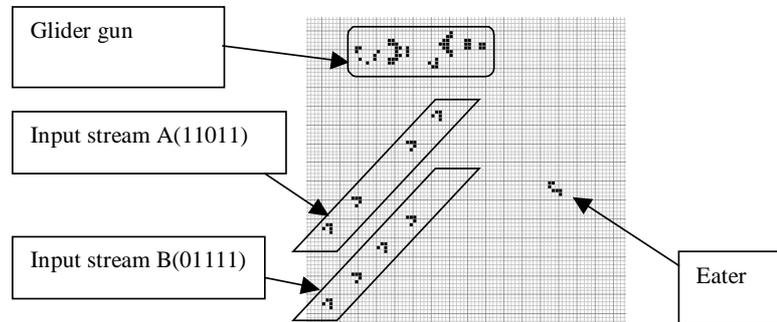


Fig. 1. An AND gate simulated by Life

In the following, we call *motif* a configuration of cells and its evolution. Then, the simulation of a logic gate by Life is possible with four motifs:

- A glider.
- A glider gun.
- A eater.
- A collision, called *vanishing*, of two streams at right angles, with the following result:
 - The destruction of the two gliders if a glider is present in each stream.
 - The survival of a glider, if a glider is present in one stream and absent in the other.

The presence of these motifs in a cellular automaton is sufficient for the possibility of a simulation of an AND gate and a NOT gate by this automaton

3. Framework

3.1. Cellular automata

Concerning this study, we explore only cellular automata with the following specifications:

- Cells have 2 possible values, 0 and 1.
- They evolve in a 2D matrix, called *universe*.
- Transition rules only take into account the eight direct neighbours of a cell for the current generation, so as to determine its states for the next generation.

We call *context of a cell* the states of the cell and its 8 neighbours. A cell thus can have 512 different contexts. A transition rule is defined as a boolean function that maps each of the 512 possible contexts to the value which will be taken by the concerned cell at the next generation. Therefore the underlying space of automata includes 2^{512} rules.

The simulation of logic circuits by Life uses a glider moving in any direction so, we choose to consider only isotropic rules. An isotropic rule is a rule in which

symmetrically equivalent contexts have the same associated value (cf. Figure 2 where 4 symmetric contexts are shown). In the following, we call "context" a group of symmetrically equivalent contexts.

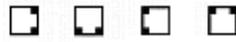


Fig. 2. Four symmetrically equivalent contexts

There exist 102 groups of symmetrically equivalent contexts. In the representation of a rule, each group is identified by one of its elements (cf. figure 3) and the associated value of each context is represented by the absence or the presence of a point at its right. This representation is used in the following.

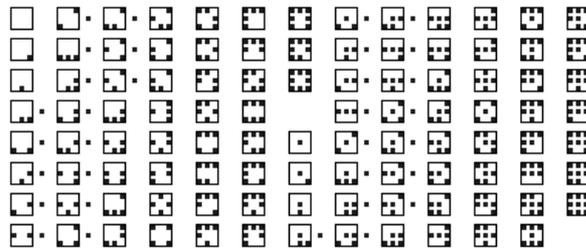


Fig. 3. Representation of a rule

We call *critical context of a motif* every context to which all the transition rules supporting this motif map the same value. For example, if a rule t supports a glider then all rules mapping the same values as t to the critical context of the glider support this glider.

3.2. Evolutionary algorithms

The evolutionary algorithm is a stochastic tool that tries to maximize a fitness function on a set of individuals called search space. A subset of the search space, population, evolves during several generations. At each generation, taking into account the fitness value of the individuals, the selection, crossover, and mutation operators produce a new population from the old one. In the two evolutionary algorithms that we present in this paper, we tried several crossover and mutation operators, several fitness functions, and several initializations of the population. For clarity, we set out only the versions of the operators and the values of the parameters which gave us the expected results.

4. A new rule

This section describes the utilization of an evolutionary algorithm for the search of new rules accepting gliders and periodic patterns.

4.1. Starting rules

Rules have been encoded by 512-bit strings in which all the bits of symmetrically equivalent contexts have the same value. The algorithm manages 50 rules produced in the following way: for each value n between 1 and 50, a string of 512 zeros is generated. In each string, n randomly chosen bits are then modified, and likewise the bits of symmetrically equivalent contexts.

4.2. Crossover and mutation

A mutation consists of modifying a randomly chosen bit, with the same weight for each of the 512 bits of a rule and likewise the bits of symmetrically equivalent contexts. Then, the isotropy of rules is kept.

We implemented a simple crossover operator at a median point.

4.3. Fitness Function

The computation of the fitness function is based on the evolution, during thirty transitions, of a "primordial soup", randomly generated in a square of 40×40 centered in a 200×200 space. During this evolution the primordial soup is the object of the following test, inspired by Bays' test [1] : each group of connected cells (see figure 4) is isolated in an empty space and evolves during 20 transitions. For each transition, the original pattern is sought in the test universe. Three cases can happen:

- The initial pattern has reappeared at its first location (it is then considered to be periodic).
- It has reappeared at another location (it is then considered to be a glider).
- It has not reappeared (it's then considered evolving).

The fitness function is evaluated as the multiplication of the number of appearances of gliders by the number of appearances of periodic patterns.

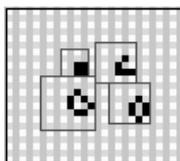


Fig. 4. Group of isolated cells

4.4. Evolutionary Algorithm

After the initialization of the 50 rules, the following cycle is iterated:

- The fitness function evaluates the rules.
- The 20 rules with the highest fitness function are kept.
- From each kept rule, a new rule is created by mutation.

- The 20 kept rules are dispatched randomly into ten couples, and from each couple a rule is created by crossover.
- A new population is created with the kept rule, the ones created by crossovers, and the ones created by mutations.

4.5. Result

This evolutionary algorithm allow us to discover rules accepting gliders. When the algorithms discovers a new rule supporting gliders, we observe a quick convergence to a population of this rule and some variants of this rule accepting the same gliders. For reasons explained in the next section, we are interested in a rule, called R (cf. figure 5), obtained by the evolutionary algorithm.

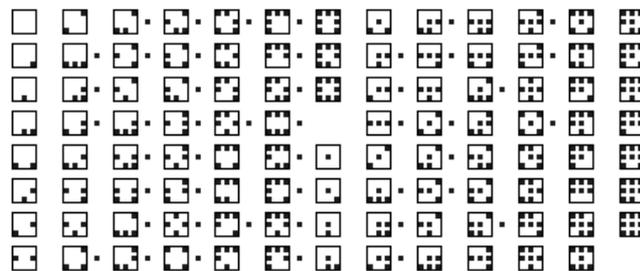


Fig. 5. Representation of rule R

5. Rule R

5.1. The glider of R

The figure 6 shows the evolution of a primordial soup under R after 100 generations of the automaton.

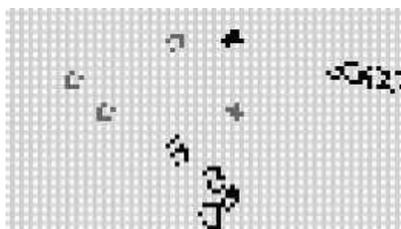


Fig. 6. Evolution of a primordial soup under R after 100 generations of the automaton (in light : a glider)

We notice the presence, in this evolution, of a glider (figure 7).

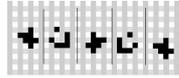


Fig. 7. Evolution of the glider of R

This glider has a period of 4 and a speed of 0,5 cells per generation

5.2. The glider gun of R

The figure 8 shows the evolution of a primordial soup under R after 100 generations of the automaton. We are interested in this rule because we notice the presence, in this evolution, of a glider gun, shown Figure 8.

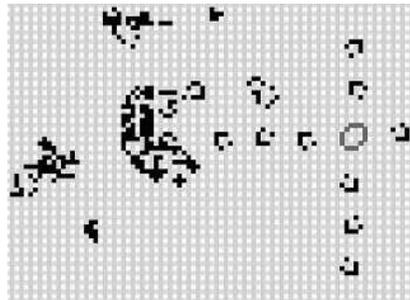


Fig. 8. Evolution of a primordial soup under R after 100 generations of the automaton (in light : a glider gun)

This glider gun emits 4 gliders in all cardinal directions every 18 generations. We wanted to know how many rules support the gun. In order to do that, we recorded the critical contexts of the gun. These are shown in black Figure 9.

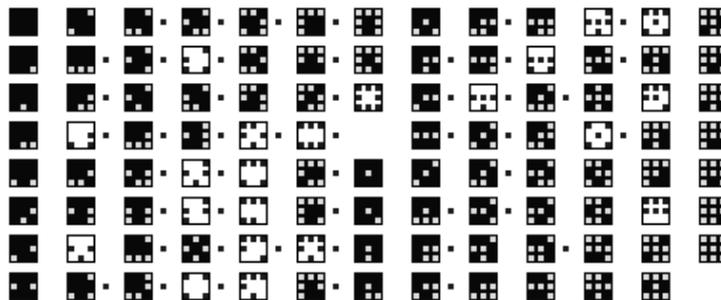


Fig. 9. R (in black : the critical contexts of the glider gun)

The gun has 81 critical contexts among the 102 existing ones, thus 2^{21} rules support the gun (and the corresponding glider) of R.

5.3. Collisions

The simulation of a logic gate by Life uses a collision of two streams of gliders. This collision must have the *vanishing* property, as described in Section 2. The critical contexts of the vanishing collision used, shown Figure 10, are critical for the glider gun, too.

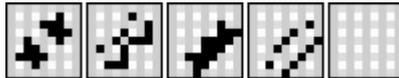


Fig. 10. Vanishing collision used of two streams of gliders

6. Eater

Inspired by [2], where the simulation of a logic gate by Life use an eater, we search, by evolutionary algorithm, an eater in R.

6.1. Population

We chose the most common periodical patterns in R to be candidate for the search of an eater (cf. Figure 11).



Fig. 11. The ten candidate patterns for the search of a eater

Figure 12 shows the position of the candidate pattern along with the stream of gliders. In this figure, the different candidate patterns for the search of a eater are in the filled square of 5*5 and this square can move in the black rectangle.

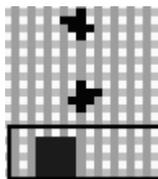


Fig. 12. Position of the candidate eater pattern along with the glider stream

An individual is defined by a triplet:

- Pattern : number, from 1 to 10, of the candidate pattern.
- Position : coordinates of the pattern compared with the glider stream.
- Rule: one of the 2^{19} rules supporting the glider gun.

The algorithm manages 50 individuals. For each individual, the pattern and the position are randomly chosen with the same weight for each of the possible choices and the rule is initialised as R.

6.2. Offspring

A mutation can have 4 variants:

- A pattern from the possible ones replaces the old one.
- The position is modified by incrementing or decrementing the ordinate.
- The position is modified by incrementing or decrementing the abscissa.
- The value mapped to a randomly chosen context, among the non-critical ones for the glider gun, is modified in the rule.

We do not use crossover in this algorithm.

6.3. Fitness function

The fitness function's goal is to evaluate the capacity of an individual to stop a glider stream. The fitness value of an individual is determined by the collision of the glider and the candidate pattern. It is equal to the number of gliders stopped by the pattern (a pattern can, for example, stop a glider but be destroyed, thus not stopping following gliders)

6.4. Evolutionary algorithm

After the initialisation of each individual, the following cycle is iterated:

- The fitness function evaluates each individual.
- The 25 rules with the highest fitness function are kept.
- From each kept rule, a new individual is created by mutation.
- A new population is created with the kept rules and the ones created by mutation.

After about ten tries of 1000 generations, we obtained a eater shown Figure 13.

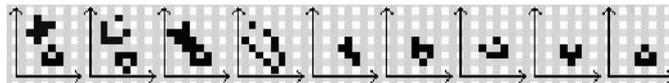


Fig. 13. Collision of a glider stream and an eater

7. Related Works

Another type of evolutionary programming, based on cellular automata, is set out in [3][4], where R. Das, M. Mitchell, and J. P. Crutchfield used genetic algorithms to evolve 1D cellular automata in order to perform computational tasks that require global information processing. In [10], genetic programming is also applied to evolve CA for random number generation.

In a previously submitted work, we discovered, by evolutionary algorithm, new 2D automata with 2 states supporting gliders. Contrary to the ones shown here, those automata are not isotropic. Still, that result contributed to demonstrating that an

evolutionary algorithm can be promising for the research of automata presenting specific behaviors.

8. Synthesis and perspectives

Based on the Conway's approach for the simulation of logic gate by Life, and using evolutionary algorithms, we identified a new automaton that is able to simulate logic gates AND and NOT. It follows from this result that genetic programming proves to be very promising for the research of 2D complex automata presenting specific behaviors.

We plan to use genetic programming for the search of the missing components for the simulation of any logic circuit, e.g. motifs allowing us to duplicate glider streams and to change their direction. Later on, our aim will be the utilization of genetic programming for the search of new universal automata.

References:

- 1 Bays C.: Candidates for the game of life in three dimensions. In *Complex Systems*, 1 (1987), 373-400.
- 2 Berlekamp E., Conway J.H, Guy R.: *Winning Ways for your mathematical plays*. Academic press, New York
- 3 Das R., Mitchell M., Crutchfield J. P. : *Evolving Cellular Automata with Genetic Algorithms: A Review of Recent Work*. In *Proceedings of the First International Conference on Evolutionary Computation and Its Applications (EvCA'96)*, Russian Academy of Sciences, 1996.
- 4 Das R., Mitchell M., Crutchfield J. P.: *The Evolutionary Design of Collective Computation in Cellular Automata*. SFI Working Paper 98-09-080.
- 5 Dytham C., Shorrocks B.: Selection, Patches and Genetic Variation: A Cellular Automata Modeling *Drosophila* Populations. In *Evolutionary Ecology*, 6 (1992) 342-351
- 6 Epstein I. R.: *Spiral Waves in Chemistry and Biology*. In *Science*, 252 (1991) 67.
- 7 Ermentrout, G. Lotti, and I. Margara , "Cellular Automata Approaches to Biological Modeling," *Journal of Theoretical Biology*, 60 (1993) 97-133
- 8 Gardner M.: *The fantastic combinations of John Conway's new solitaire game « Life »*, In *Scientific American*, (1970)
- 9 Gardner M.: *On Cellular Automata, Self-reproduction, the Garden of Eden, and the Game of Life*. In *Scientific American*, 224 (1971) 112-118
- 10 Hordijk W., Crutchfield J. P. , Mitchell M.: *Mechanisms of Emergent Computation in Cellular Automata.*, *Parallel Problem Solving from Nature-V*, 613-622, Springer-Verlag, 1998.
- 11 Wolfram S.: *Universality and complexity in cellular automata*. In *Physica D*, 10 (1984) 1-35.