Emerging Stable Configurations in Cellular Automata

Mattia Vinci and Roberto Micalizio

Dipartimento di Informatica
Università di Torino
corso Svizzera 185, 10149 Torino
mattia.vinci.1@gmail.com,micalizio@di.unito.it

Abstract. This paper considers how simple and limited entities, such as Cellular Automata (CA), can organize themselves giving rise to stable and complex behaviors. We begin by introducing Cellular Automata, simple discrete computational models useful to describe the evolution of a system following simple rules. Then we propose a possible architecture to implement a class of CA onto a Unix system. In particular, we propose a distributed solution where a number of cooperating processes cooperate in order to evolve the current configuration of the Universe.

Keywords: Cellular Automation, Self-stabilization, Conway's Game of Life

1 Introduction

Distributed and multiagent systems are nowadays common in many aspects of our daily life. From financial agents selling and buying stocks, to swarms of UAVs flying in formation, there are many scenarios in which heterogeneous agents interact with one another in order to accomplish their own goals. In principle, one can think of these systems, or societies, as composed by a number of single entities, so that the behavior of the society as a whole can be represented just in terms of the behaviors of its entities. Unfortunately, such a simplistic view does not take into account that complex behaviors can emerge within an agent society even though these behaviors have not been designed. In other words, an agent society can exhibit an organized behavior even though no (external) agent organizes it. A simple, yet effective, way to study emerging behaviors consists in the adoption of Cellular Automata (CA). CA are simple computational models represented as matrices, in which each cell is a simple Finite-State Machine, often with just two possible states: *alive* and *dead*. It is worth noting that the initial conditions of a CA system satisfy the ones defining a *chaotic system*:

a. sensitive to initial conditions
b. topologically mixing
c. having dense periodic orbits

where point *a.* means that an arbitrarily small perturbation of the current trajectory may lead to significantly different future behaviour - what is usually known as *the Butterfly Effect*; being *topologically mixing* means evolving over time so that any given region or open set of its phase space will eventually overlap with any other given region. Point *c.* states that every point in the space is approached arbitrarily closely by periodic orbits.
Although these premises would suggest that a CA could evolve in a chaotic way, it has been demonstrated in many practical scenarios that the simple rules governing the states of the cells do have an impact on the system as a whole, and in particular, can give rise to a simple form of self-organization. In this paper, we start by giving a description of the functioning of a Cellular Automata and in particular of the "Game of Life", and then present a possible architecture for implementing a CA system. Ideally, since each cell in the system is autonomous (i.e., it is a simple agent), it should be implemented as an independent process, or thread. For scalability reasons, however, we propose an architecture where a number of cooperating processes, named *Evolving Processes*, coordinate with one another in order to evolve consistently the entire system.

2 Background

2.1 Cellular Automata

A *Cellular Automata (CA)* system is a computing model in which the environment - usually referred to as *Universe* - is a grid of cells extending in a finite number of dimensions. Each cell can be in one of a finite number of states.
At each step of the universe evolution, generally called a *generation*, each cell evolves into a state that is a function of the previous cell's neighborhood states. The set of neighbouring cells can be defined in several ways, the most common being the *Von Neumann set* (the set of perpendicularly adjacent cells) and the *Moore set* (all adjacent cells).
The evolution rules can be expressed in several ways, the most common being the *Wolfram* code (by Wolfram & Packard, 1985) and *Mcell* (Mirek Wjtowicz, 1999). The Wolfram code is a decimal number whose binary represantation is a byte that describes, for each possible alive neighbouring set, the state the cell should be in.
The MCell representation encodes the same information in a string of the type Sx/By, in which x is a list of numbers of alive neighbours required for the Cell to survive, and y is the list of numbers of alive neighbours required for a dead cell to become alive.
The evolution of a CA can follow different path; based on their behaviours in time, CA are usually classified, following Stephen Wolfram [1], in four main categories:

- *Uniformity* automata in which patterns generally stabilize into homogenity
- *Repetition* automata in which patterns evolve into mostly stable or oscillating structures

- *Random* automata in which patterns evolve in a seemingly chaotic fashion
- *Complexity* automata in which patterns become extremely complex and may last for a long time, with stable local structures

The interests on CA is due to the fact that they can be set to model real-life situations. For instance, the emerging of fractal structures in many real-world shapes (e.g., trees, shells, and crystals) is easily emulated by many simple CA. These models have also been used to simulate more complex situations, such as social and economic processes [2], the evolution of bacteria colonies [3], urban processes [4] or to provide simpler models of common differential equations of physics, such as the heat and wave equations and the fluid equations. Moreover, using cells of different shape allows to construct even more structures; using hexagonal CA with the rule "alive cells die if only one cell in its neighborhood is dead" is possible to obtain the *von Koch snowflake* [5] and other crystalline structures.

2.2 Conway's Game of Life

In the 1970s the British mathematician Conway, looking for rules that lead to interesting behaviors, introduced the rule *S23/B3*.
This rule states that any cell, if alive, can survive only if it has exactly 2 or 3 alive neighbors; otherwise it dies of "loneliness" or "overcrowding". A dead cell can come alive if it has exactly 3 alive neighbors.
This rule presents a behavior that at first sight can look similar to evolving patterns found in real biological world, and because of this is widely known as *the Game of Life*.
The most interesting feature of this rule is its natural tendency to self-organization: given a random initial condition, the Game may evolve forming stable small structures that can be:

- *still life* if they reach a stable still state
- *oscillators* if they reach a set of repeating states
- *spaceships*, structures that move themselves around the grid

It has also been demonstrated that *Life*-like CA can implement logic gates, realize arithmetic tasks, and simulate Finite State Machines. Indeed, Wolfram [1] has demonstrated that a cellular automata system, when behaving under a specific rule, is Turing-complete.

3 Architecture and Implementation

We realized a Cellular Automata system as a part of a laboratory project on Unix operating system. It must therefore be noticed that the main purpose of the project was not AI, but how to use the Inter-Process Communication facilities offered by Unix.

Of course, it is not hard to see that concurrent processes can be seen as agents. In addition, processes using semaphores and shared memories, can be seen as agents competing for accessing the resources, but at the same time, these agents could cooperate in order to reach a common goal. After these simple premises in mind, we implemented the CA system by means of four types of processes (i.e., agents):

- Memory Manager
- Evolving Agents
- Universe Displayer
- User Menu

The Memory Manager is the main agent of our implementation, it allocates in a segment of shared memory a matrix of `short` representing the Universe: an alive cell has value 1, a dead one has value 0; the default size of the universe is 130x43 cells.
The synchronization of all the agents, as well as the consistency of the displayed configuration, are assured by a pool of semaphores, allocated by the Manager, that also guarantees mutually exclusive access to the shared memory. The Manager also allocates three queues to allow the communication among the agents.
The default number of *Evolving Agents* is 5. Each of these agents asks the Memory Manager the permission to access the shared memory; if granted, the Manager inserts the process information into a list of currently active agents, and sends back a message containing the identifiers of the shared memory and of the semaphores together with the portion of the area assigned to that agent.
The evolution step requires that each Evolving Process copies its portion of shared memory into a local array. The local array is used to represent the current state of universe, in a way that the Evolving Agent can apply the evolution rules reading from its local array and modifying accordingly its segment of the shared matrix. When all the Evolving Agents have terminated an evolution step, the shared matrix will contain the new configuration. Counter Semaphores are used in order to synchronize the processes so that no one starts a new evolution step if some other process has not completed its previous one.
The *Displayer*, is now activated to show the current state of the universe in a terminal window using the *ncurses* library. To simplify the interface, we just use a matrix of characters putting a character '$\diamond$' where the corresponding cell is alive, and leaving the spot blank otherwise. In this case too, a semaphore is used to guarantee that a new evolution step is started only after the display of the current state has been completed.
The interaction with the user is allowed by means of an interactive menu (*User Menu* process), from which the user can select the type of automata and the rules to simulate.
The user can choose to randomly initialize the universe and evolving it following one of the hard-wired rules:

- Game of Life (S23 / B3)
- 34 Life (S34 / B34)
- High Life (S23 / B36)
- Day and Night (S34678 / B3678)

– Sierpinski Triangle (1D Xor based rule)

or she can select to load both initial pattern and evolution rule from a file.

The user can also interact with the *Universe* using the mouse to turn on and off single cells in order to see how slightly different initial conditions can give rise to very different evolutions, and compare them to see if and how they stabilize, and how long (how many evolution steps) it took.
At any time the user can interrupt the evolution loop using system signals. In addition, if activated, a mechanism to detect short periodic loops re-initializes the universe when overall patterns are closely repeating.

The entire code of this project is shared through GitHub. [10]

4 Characterizing Stable Configurations

Although the developed system is still in its early stage, it is important to note that it could be a useful tools to spread some basic ideas of Artificial Intelligence. As we have already noticed, many real-world systems, especially biological ecosystems, present stable or ordered configurations that emerge autonomously without the presence of a coordinator.

Since stable configurations are recurrent in CA systems too, our purpose is to characterize initial configurations and evolution rules. More precisely, our idea is to start from a stable configuration, and allow the user to perturb this initial configuration by adding or deleting alive cells. This change obviously creates an instable situation, that will eventually evolve in a new stable configuration. Such a kind of interaction can be useful for a user to understand which initial configurations evolve more rapidly into new stable configurations and with which rules. Moreover, even a non-expert user can easily understand how her/his changes impact on the whole Universe. The prototype could therefore be used as an educational tool for teaching young students the well-known "butterfly effect". In fact, it could be used as a game where the user has to change the world by adding/removing a specific number of alive cells, but the impacts of her/his changes should be as limited as possible. In other words, the new stable configuration obtained after the changes should be as similar as possible to the original one. To assess the similarity between two configurations different measures could be devised. For instance, the number of alive cells of the new stable configuration should be the same, or close, to that of the original one. Also the number of iterations required for evolving to the new stable configuration could be an interesting parameter to be studied. For example, a stable configuration C_1 could be preferred to another configuration C_2, if C_1 has been reached with a minor number of evolutions than C_2.

5 Conclusions

We have used Cellular Automation as a mean to observe evolution of seemingly chaotic systems into organized complex structures.

We presented a possible architecture to implement such automata onto a Unix system, using the operating system IPC structures (message queues, semaphores, signals, shared memory), various cooperating synchronized processes, and continuously displaying the evolution state into a terminal window.
Repeatedly observing the state of the system after a certain amount of time we noticed how the initial chaos evolved itself into a complex stable structure, as if the global equilibrium were achieved through the application of strictly local rules. This makes us conclude that it is possible for initially chaotic configurations to self stabilize and give rise to more complex entities by following local simple evolution rules.

This work represents just a first step in the study of CA. As a future work, we are interested to study how CA combined with Genetic Algorithms can simulate sophisticated real-world domains.

References

1. Wolfram, Stephen. A new kind of science. Vol. 5. Champaign: Wolfram media, 2002.
2. Bagnoli, Franci, Rechtman *Chaos in a simple cellular automaton model of a uniform society*, LNCS 3305, pp. 513-522, 2004.
3. Krawczyk1, Dzwinel, David A.Yuen, *Nonlinear Development of Bacterial Colony Modeled with Cellular Automata and Agent Objects*, International Journal of Modern Physics C 14.10 (2003): 1385-1404.
4. Santé, Inés and García, Andrés M and Miranda, David and Crecente, Rafael, "Cellular automata models for the simulation of real-world urban processes: A review and analysis." Landscape and Urban Planning 96.2 (2010): 108-122.
5. Gravner, Janko, and David Griffeath. "Modeling snow crystal growth I: Rigorous results for Packard's digital snowflakes." Experimental Mathematics 15.4 (2006): 421-444.
6. Rendell Attic *http://rendell-attic.org*
7. JSLife patterns *http://entropymine.com/jason/life*
8. Mirek Cellebration *http://www.mirekw.com/*
9. SUCS http://sucs.org/ pwb/report/
10. Code: https://github.com/sowdust/Cell_Auto.git