



**HAL**  
open science

# Organic Structures Emerging From Bio-Inspired Graph-Rewriting Automata

Paul Cousin, Aude Maignan

► **To cite this version:**

Paul Cousin, Aude Maignan. Organic Structures Emerging From Bio-Inspired Graph-Rewriting Automata. 24th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2022), Sep 2022, Linz, Austria. pp.293-296, 10.1109/SYNASC57785.2022.00053 . hal-03869492

**HAL Id: hal-03869492**

**<https://hal.science/hal-03869492v1>**

Submitted on 24 Nov 2022

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Organic Structures Emerging From Bio-Inspired Graph-Rewriting Automata

Paul Cousin and Aude Maignan

Univ. Grenoble Alpes, CNRS, Grenoble INP, LJK  
Grenoble, France

## Abstract

Graph-Rewriting Automata (GRA) are an extension of Cellular Automata to a dynamic structure using local graph-rewriting rules. This work introduces linear algebra based tools that allow for a practical investigation of their behavior in deeply extended time scales. A natural subset of GRA is explored in different ways thereby demonstrating the benefits of this method. Some elements of the subset were discovered to create chaotic patterns of growth and others to generate organic-looking graph structures. These phenomena suggest a strong relevance of GRA in the modeling natural complex systems. The approach presented here can be easily adapted to a wide range of GRA beyond the chosen subset.

*Keywords* — cellular automata, graph-rewriting automata, dynamical systems, artificial life, complexity.

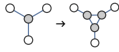
## 1 Introduction

The idea of extending the study of discrete dynamical systems like cellular automata to systems with evolving typologies has already been approached in several ways. DEM-Systems [1–3], Structurally Dynamic Cellular Automata [4–6], Generative Network Automata [7–9] and Graph-Rewriting Automata [10–12] are related concepts sharing this goal. These systems can produce a range of new behaviors compared to their fixed topology counterparts, however their implementation is much less straightforward.

In this paper, new linear algebra based tools will be presented, then applied to study a natural subset of GRA. Links to the Mathematica source code, a more recent GPU-accelerated Python implementation, and more information can be found at [paulcousin.github.io/graph-rewriting-automata](https://paulcousin.github.io/graph-rewriting-automata).

A GRA consists of an initial graph  $G_0$  defined at time step  $t = 0$  and a rule to iteratively evolve it to any discrete time step  $t > 0$ .  $G_t$  is the graph obtained at time  $t$ . The studied subset of GRA will be defined by the following restrictions:

1. 3-regular, undirected and finite graphs,
2. binary labeling of vertices  $v$  with two possible states  $s$ :
  - $s(v) = 1$ , called “alive” and colored purple ●,
  - $s(v) = 0$ , called “dead” and colored orange ●,
3. rules local to a vertex and its adjacent vertices, deterministic and applied simultaneously on the entire graph,
4. only one type of topology altering operation called division.

Divisions will be operated as follows: 

In this model, vertices can, for example, be seen as cells that divide under specific local conditions (internal and environmental). An initially simple graph can thus grow into a large organism with a complex self-organized structure.

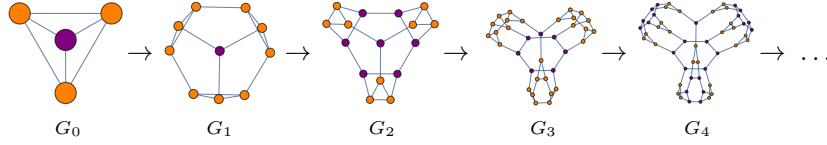


Figure 1: Example of evolution starting from a simple initial graph.

## 2 Graphs and Rules

A **graph**  $G$  is fully described by an **adjacency matrix**  $\mathcal{A}$  and a **state vector**  $\mathcal{S}$ :  $G_t = \{ \mathcal{A}_t, \mathcal{S}_t \}$

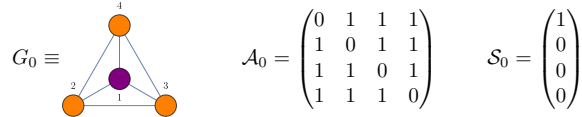


Figure 2: Example of the graph  $G_0$  from *Figure 1*.

The **neighborhood**  $N(v)$  of a **vertex**  $v$  is the set of vertices which are adjacent to it. The **state**  $s$  of a vertex being either dead or alive and there being from 0 to 3 alive vertices in its neighborhood, there are only 8 possible **configurations**  $c$ , which can be ordered in this way:

$$c(v) = 4 \times s(v) + \sum_{i \in N(v)} s(i) \quad (1)$$

For instance, if  $v$  is a dead vertex surrounded by 3 dead vertices,  $c(v)$  will be equal to 0 (see *Figure 3* for all the other configurations).

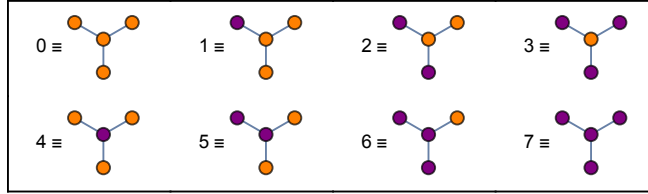


Figure 3: Eight possible configurations.

The space of possible rules applicable in the subset of GRA defined in *Section 1* is finite. Every rule must specify, for each configuration, whether the vertex will be alive or dead at  $t + 1$  and whether it will have undergone a division, leading to 4 possible final states. Thus, there are  $4^8 = 65,536$  possible rules. Each rule can be described by two functions:

$$R : [[0, 7]] \rightarrow \{ 0 \equiv \text{orange}, 1 \equiv \text{purple} \} \quad (2)$$

$$R(c(v_t)) = s(v_{t+1})$$

$$R' : [[0, 7]] \rightarrow \{ 0 \equiv \text{no division}, 1 \equiv \text{division} \} \quad (3)$$

$$R'(c(v_t)) = \begin{cases} 1, & \text{if } c \text{ leads to a division at } t+1 \\ 0, & \text{otherwise} \end{cases}$$

Every rule can thus be labeled by a unique **rule number**  $n$ .

$$n = \sum_{i=0}^7 \left[ 2^i R(i) + 2^{i+8} R'(i) \right] \quad (4)$$

This labeling system, inspired by the Wolfram code [13], is such that a rule number in its binary form displays the behavior of the rule. Starting from the right, the 8 first digits indicate the future state for each configuration as they have been ordered previously. The 8 following digits show when a division occurs.

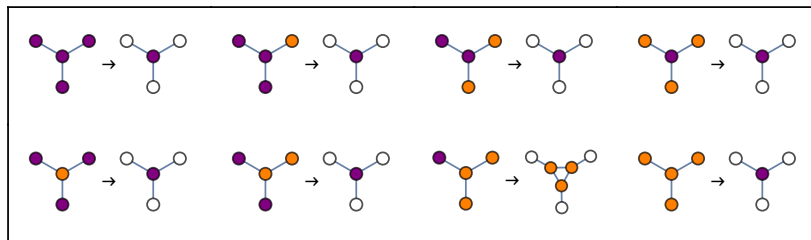


Figure 4: Rule of the example in *Figure 1* ( $n = 765 = 10,1111,1101_2$ ).

### 3 Implementation

Implementing GRA as described below must leverage a sparse array format to be efficient.

The application of a rule to a graph will come in several steps which are strictly equivalent to the rule being applied all at once.  $o$  being the order of the graph and  $@$  being the operator applying a function to every element of a vector, the first step consists of computing a **configuration vector**  $\mathcal{C}$ .

$$\mathcal{C} = \begin{pmatrix} c(v_1) \\ \vdots \\ c(v_o) \end{pmatrix} = 4 \times \mathcal{S} + \mathcal{A} \cdot \mathcal{S} \quad (5)$$

$\mathcal{S}$  can then be updated as follows.

$$\mathcal{S} = R @ \mathcal{C} \quad (6)$$

A **division vector**  $\mathcal{D}$  is computed similarly.

$$\mathcal{D} = R' @ \mathcal{C} \quad (7)$$

Divisions can now be performed one by one as a combination of simple operations on matrices. The first 1 in  $\mathcal{D}$  signals the vertex to divide. For the state vector, suffice to triple the line corresponding to this vertex. For the adjacency matrix, both the line and the column must be tripled. Ones then have to be spread across these lines and columns and the intersection must be filled with a sub-matrix containing zeros on the diagonal and ones everywhere else. Finally, the first 1 in the division vector is turned into a 0 and then tripled. This process must be repeated until  $\mathcal{D}$  is a null vector.

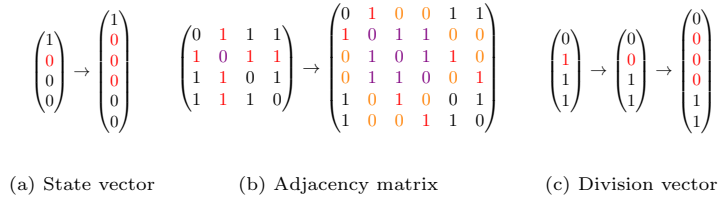


Figure 5: Division operated on the second vertex of the graph from *Figure 2*.

### 4 Exploration

An exhaustive exploration will be performed on the set of rules with exactly one division. This set being color symmetric, if it is the case as well for  $G_0$ , considering the half of it were the division is applied to a dead cell will give the full picture. This makes a total of 1024 rules to be explored.

$$n \in \{i + 2^j \mid i \in [[0, 255]], j \in [[8, 11]]\} \quad (8)$$

To investigate the behavior of these rules, a simple graph  $G_0$  depicted in *Figure 6* and containing all 8 configurations will serve as a starting point. This ensures that  $G_1 \neq G_0$ .

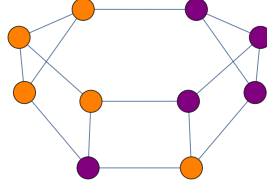


Figure 6:  $G_0$ : Initial graph with all 8 configurations.

All the results and plots in this paper were computed in Mathematica 13.0 [14], on a MacBook Pro 2020 with an Intel Core i5 Quad-Core 2.0GHz and 16GB of memory. Graphs are displayed by the default graph plotting function in Mathematica and can be better seen by enlarging this pdf.

## 5 Growth

One natural way to classify rules is by observing the pattern of growth they produce. *Table 1* shows a classification obtained using several methods, mainly the least squares method.

Table 1: Number Of Rules By Growth Pattern.

Halted	Linear			Quadratic	Exponential	Unclassified
	strict	periodic	chaotic			
422	73	19	3	1	374	132

### 5.1 Halted growth

A total of 422 rules lead to a halt when reaching a cycle. These cycles have periods of 1, 2, 3, 6, or 8.

Table 2: Period distribution in rules leading to cycles.

Period	1	2	3	6	8
Number of rules	310	102	2	6	2

### 5.2 Linear growth

73 rules produce a strictly linear growth, with the same number of vertices gained at each time step. 19 rules create a repeating pattern of growth that averages to linearity. The largest period spanned 42 time steps and was produced by rule 2183. This comes from the two main structures of the graph: a loop growing in 7 steps  $\{2, 0, 0, 2, 0, 0, 0\}$  and a braid growing in 6 steps  $\{4, 0, 0, 4, 2, 0\}$ .

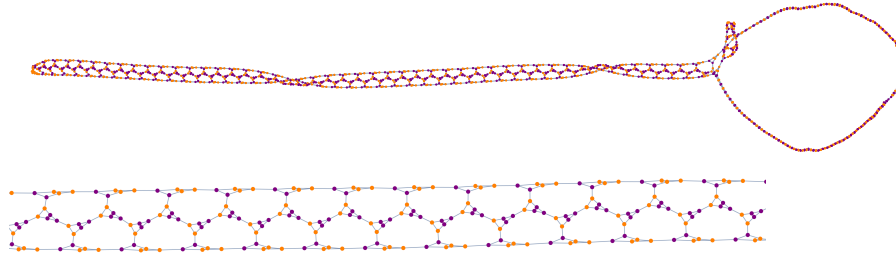


Figure 7: Rule 2183 at  $t = 600$  along with a close-up of the braid.

More surprising though are the 3 rules giving rise to a chaotic linear growth: 2222, 2238 and 2239. These rules produce ladder-like loops in which states change according to an Elementary Cellular Automaton rule [15]. For the sake of concision, only rule 2222 will be presented.

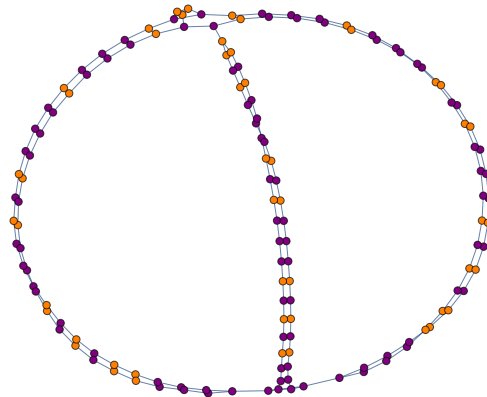


Figure 8: Rule 2222 at  $t = 200$ .

This rule, although producing a rather simple graph structure, is interesting by the rhythm of its growth. At any time step, it will either grow of 0, 2, 4 or 6 new vertices which averages to a linear growth. It can be noted that there are no increase of 6 vertices at once for  $t \in [[1217, 20608]]$ .

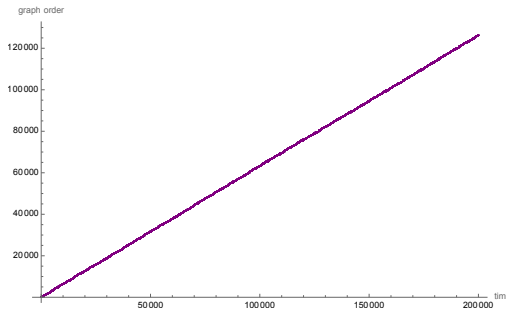


Figure 9: Rule 2222: linear growth.

Model fit: $o = 72.8095 + 0.630684 \times t$	Adjusted $R^2$ : 0.999992
--	---------------------------

What is remarkable here is the fact that this growth does not follow a repeating pattern but, instead, seems to have chaotic properties. To illuminate the peculiar aspect of this phenomenon, *Figure 10* shows the distribution of the lengths of time intervals without growth in a log-linear plot.

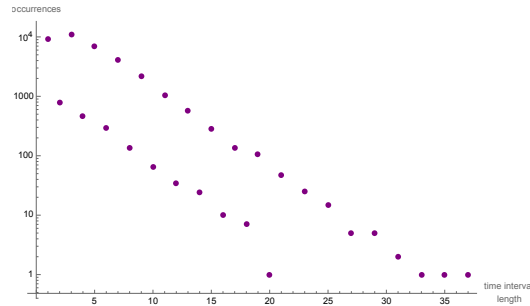


Figure 10: Rule 2222: length distribution of time intervals without growth computed for  $0 \leq t \leq 200,000$ .

### 5.3 Quadratic growth

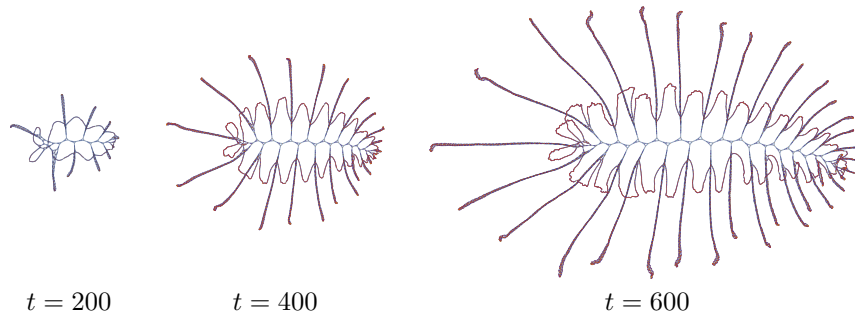


Figure 11: Rule 2182.



Rule 2182 was found to be unique in the set regarding its growth. It produces a particular structure that grows almost quadratically, the exponent being slightly above 2.

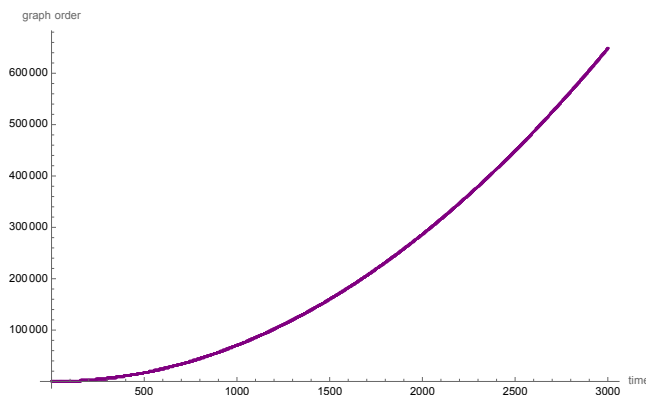


Figure 12: Rule 2182: quasi-quadratic growth.

Model fit: $o = 0.0620341 \times t^{2.01874}$	Adjusted $R^2$ : 0.999999
---	---------------------------

## 5.4 Exponential growth

Rule 256 produces a perfect exponential growth from  $t = 1$ . Among the 373 other exponential rules, some display simple fractal structures, others are more complex.

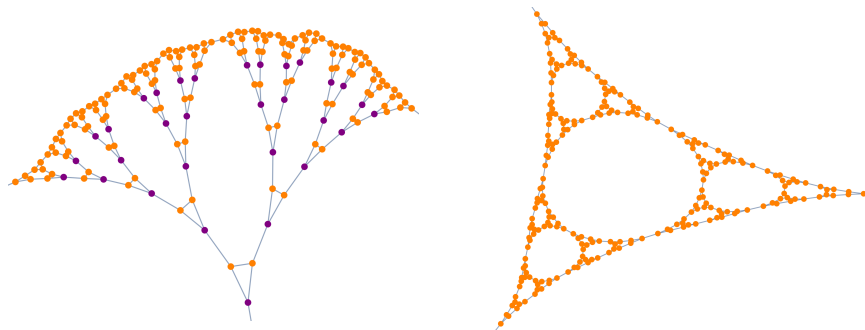


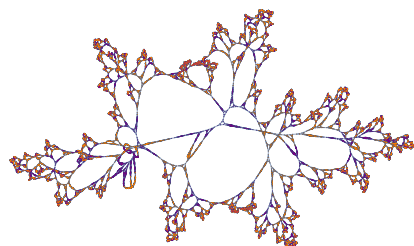
Figure 13: Examples of fractal structures found in exponential rules.

## 5.5 Unclassified

132 rules have not allowed for a confident classification. Suspecting interesting behaviors to be responsible for these less trivial growth patterns, all of them were observed individually and a selection of rules producing remarkable graph structures is presented in *Section 6*.

## 6 Graph structures

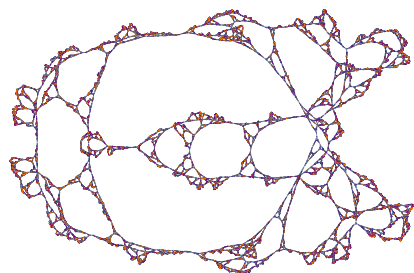
As *Figure 11* already hinted at, some rules create organic-looking graph structures. *Figure 14* shows a selection of graphs with this property, labeled with the number of the rule that produced them. These structures are particularly evocative. Some even seem familiar, resembling macroscopic algae or related photosynthetic eukaryotes.



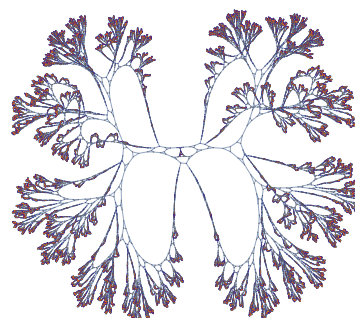
549



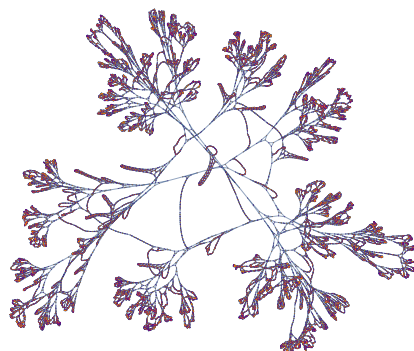
618



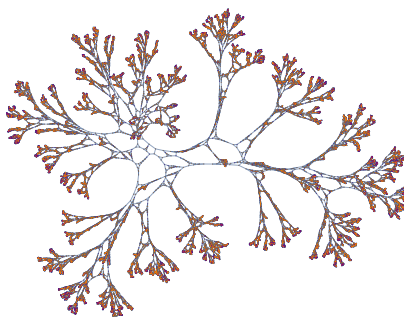
1062



1111



2199



2236

Figure 14: Some organic-looking graph structures.

## 7 Conclusion

In this paper, a new approach to Graph-Rewriting Automata based on linear algebra was presented. When leveraging a sparse array format, it can be used to implement GRA much more efficiently than with previous methods. To demonstrate the benefits of this approach, a natural subset of GRA was explored only using a MacBook Pro 2020.

A focus was first put on the evolution of the number of vertices over time, leading to a classification of rules by growth pattern. Several noteworthy phenomena were discovered through this approach. It also led to a category of unclassified growths in which interesting behaviors were suspected. The graphs of this category were thus visually inspected on an individual basis and some remarkable structures were discovered.

In future work, the ideas presented here could easily be adapted to cover a variety of other types of GRA. Here are a few examples of possible trajectories:

- using continuous valued or non-binary discrete states,
- working with  $d$ -regular or non-regular graphs,
- reaching an extended neighborhood with powers of  $\mathcal{A}$ ,
- working with directed graphs using  $\mathcal{A} \cdot \mathcal{S}$  and  $\mathcal{A}^T \cdot \mathcal{S}$ ,
- including other topology altering operations.

Other tools to analyse graphs directly from their adjacency matrix and state vector could also be developed to gain new insights into the behaviors and properties of GRA.

## Acknowledgment

This work was partially supported by the Université Grenoble Alpes, through an Excellence Internship which took place between June 7<sup>th</sup> and July 15<sup>th</sup> 2022.

## References

- [1] R. Edwards and A. Maignan, “A class of discrete dynamical systems with properties of both cellular automata and l-systems,” *Natural Computing*, vol. 19, no. 3, pp. 609–641, 2020.
- [2] —, “Dem-systems: a new type of adaptive system,” *Exploratory papers of automata*, 2016.
- [3] —, “Complex self-reproducing systems,” in *ISCS 2013: Interdisciplinary Symposium on Complex Systems*. Springer, 2014, pp. 65–76.
- [4] A. Ilachinski and P. Halpern, “Structurally dynamic cellular automata.” 2009.
- [5] R. Alonso-Sanz and M. Martín, “A structurally dynamic cellular automaton with memory in the hexagonal tessellation,” in *International Conference on Cellular Automata*. Springer, 2006, pp. 30–40.

- [6] R. Alonso-Sanz, “A structurally dynamic cellular automaton with memory,” *Chaos, Solitons & Fractals*, vol. 32, no. 4, pp. 1285–1295, 2007.
- [7] H. Sayama, “Generative network automata: A generalized framework for modeling complex dynamical systems with autonomously varying topologies,” in *2007 IEEE Symposium on Artificial Life*. IEEE, 2007, pp. 214–221.
- [8] H. Sayama and C. Laramee, “Generative network automata: A generalized framework for modeling adaptive network dynamics using graph rewritings,” in *Adaptive networks*. Springer, 2009, pp. 311–332.
- [9] J. Schmidt and H. Sayama, “Designing and evaluating algorithms for automated discovery of adaptive network models based on generative network automata,” in *2013 IEEE Symposium on Artificial Life (ALife)*. IEEE, 2013, pp. 27–34.
- [10] K. Tomita, S. Murata, and H. Kurokawa, “Self-description for construction and computation on graph-rewriting automata,” *Artificial Life*, vol. 13, no. 4, pp. 383–396, 2007.
- [11] —, “Asynchronous graph-rewriting automata and simulation of synchronous execution,” in *European Conference on Artificial Life*. Springer, 2007, pp. 865–875.
- [12] K. Tomita, H. Kurokawa, and S. Murata, “Graph-rewriting automata as a natural extension of cellular automata,” in *Adaptive Networks*. Springer, 2009, pp. 291–309.
- [13] S. Wolfram, *A New Kind of Science*. Wolfram Media, 2002.
- [14] W. R. Inc., “Mathematica, Version 13.0,” champaign, IL, 2021.
- [15] E. W. Weisstein, “Elementary cellular automaton,” *mathworld.wolfram.com*, 2002.