Searching for complexity in cellular automata

Giordano Colombo, Andrea Del Torchio

July 16, 2024

Abstract

The study of damage spreading in cellular automata (CA) is essential for understanding chaos and phase transitions in CA and complex systems in general.
Using the concept of the Boolean derivative the damage spreading for one-dimensional elementary cellular automaton is studied, defining the Lyapunov exponents that allow for the study of Lyapunov spectra of ECA. The spectra found for II class rules 26, 108, and 127 are compatible with theoretical expectations.

1 Introduction

Cellular automata can be described as artificial programmable "universes", which are discreet in time and space. The physics of these logical universes is deterministic and local. Local means that the state of a cell at time $t+1$ is only a function of its state and the states of the cells in a certain neighborhood at time t. Deterministic means that once a local physics and an initial state of a cellular automaton has been chosen, its future evolution is uniquely determined.
The most interesting cellular automata, balancing chaoticity and tendency to generate structures, show complex behavior that cannot be easily predicted [1]. How to define complexity is not a trivial question. Which universal features are shared by apparently different complex systems? The behavior of physical systems close to critical points may answer this question. It is well known, from the theory of phase transitions, that a given system (possibly made of many subsystems) can undergo strong qualitative changes in its macroscopic properties if a suitable control parameter is adequately tuned. Close to these critical points, some key characteristic constants (the so-called critical exponents) are the same for very different systems [2]. At critical points, fractal structures and complex dynamical patterns appear spontaneously. Observing such properties in those systems that are called "complex," one can conjecture that complexity tends to appear close to instability points.
The study of perturbation or defects spreading in cellular automata, is essential for understanding when these systems show complex behavior. It also helps us to understand the dynamics and emergent properties of complex systems in general.
In this paper an analysis of damage spreading in cellular automata is proposed. It is made use of the boolean derivative which allows us to compute the Lyapunov spectra of cellular automata. This approach parallels the well-established method used in continuous-state dynamical systems, facilitating the application of established theorems from dynamical systems theory.

2 Preliminary definitions

2.1 Cellular Automata

It is useful to first define a one-dimensional CA.
One-dimensional CA is formally defined by a quadruplet $\{\mathbf{f}, \mathbb{S}, \mathbb{Z}, \mathrm{N}\}$, where

$$\mathbf{f}\colon \mathbb{S}^{\mathbb{Z}} \to \mathbb{S}^{\mathbb{Z}}$$

is a map of global evolution of the system, the set $\mathbb{S} = \{s_i\}_{i=0}^{k-1}$ is the space state, $\mathbb{Z}$ represents the infinite one-dimensional "lattice" and $N(x_i)$ is a neighborhood model, e.g. $N(x_i) = \{x_{i-1}, x_i, x_{i+1}\}$ is a nearest neighbors model.
The map of global evolution $\mathbf{f}$ is defined as

$$\mathbf{f}(\mathbf{x}^t) = \left\{\phi(x_i^t)\right\}_{i\in\mathbb{Z}} = \left\{x_i^{t+1}\right\}_{i\in\mathbb{Z}}$$

where $\mathbf{x}^t = \{x_i^t\}_{i\in\mathbb{Z}}$ is the global configuration of the system at time step t, dependent on the state of individual cells at the same time step, and ϕ is a local transition function also called "rule" of the CA. The state of a cell at time t can be obtained by repeatedly applying the rule to the initial state: $\phi^t(x_i^0) = x_i^t$
The most basic CA have a binary state space $\mathbb{S} = \{0, 1\}$ and a nearest neighbors model $N(x_i) = \{x_{i-1}, x_i, x_{i+1}\}$ and are called elementary CA (ECA).

2.2 Standard definition of Lyapunov exponents

The Lyapunov spectrum of a discrete-time continuous-state dynamical system is next recalled.
Consider an n-dimensional dynamical system on the configuration space $\mathbb{R}^n$ defined by: $\mathbf{x}^{t+1} = \mathbf{f}(\mathbf{x}^t)$, $\mathbf{x} \in \mathbb{R}^n$. An infinitesimally small perturbation, denoted $\delta\mathbf{x}^0$, to the initial value $\mathbf{x}^0$ will cause a change in the trajectory of $\mathbf{x}^0$. One may study how the two initially close trajectories converge or diverge in the configuration space $\mathbb{R}^n$ due to such an initial perturbation by tracking the difference $\Delta\mathbf{x}^t = \mathbf{f}^t(\mathbf{x}^0) - \mathbf{f}^t(\mathbf{x}^0 + \delta\mathbf{x}^0)$. Convergence indicates stability while exponential divergence indicates unstable or chaotic dynamics. To the first order one can write $\Delta\mathbf{x}^t = M\delta\mathbf{x}^0$ where M is a matrix n×n that can be obtained by applying t-times the jacobian matrix

$$J_{\mathbf{f}}(\mathbf{x}^i) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}\Big|_i & \cdots & \frac{\partial f_1}{\partial x_n}\Big|_i \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}\Big|_i & \cdots & \frac{\partial f_n}{\partial x_n}\Big|_i \end{bmatrix}.$$

to the initial perturbation $\delta\mathbf{x}^0$ i.e.

$$\Delta\mathbf{x}^t = J_{\mathbf{f}}(\mathbf{x}^{t-1}) \cdots J_{\mathbf{f}}(\mathbf{x}^0)\Delta\mathbf{x}^0.$$

Consider the matrix $B = M^T M$ which is defined as the product of the matrix M and its transpose. This matrix will have real non-negative eigenvalues, whose square roots coincide with the singular values Λ_i of the matrix M. The Lyapunov coefficients are then defined as follows:

$$\lambda_i(\mathbf{x}^0, t) = \frac{1}{t} log(\Lambda_i(\mathbf{x}^0, t)).$$

If $\lambda_i < 0 \forall i = 1, \ldots, n$ there will be reduced sensitivity to initial conditions, and thus the system evolves quickly towards stability. Otherwise, if there $\exists i$ t.c. $\lambda_i > 0$ There will be a high sensitivity to initial conditions, resulting in unpredictable and chaotic behavior. Note that different initial configurations can lead to different trajectories, and consequently, the computed Lyapunov spectrum may differ for every attractor there is an associated Lyapunov spectrum. As such, the initial configuration can affect the Lyapunov spectrum, since it determines on which attractor the system settles.

2.3 Damage spreading in CA

A finite one-dimensional CA of size n can be viewed as an n-dimensional discrete-time discrete-state dynamical system: $\mathbf{x}^{t+1} = \mathbf{f}(\mathbf{x}^t)$, $\mathbf{x} \in \mathbb{S}^n$,
Since a CA is an n-dimensional dynamical system, one approach is to use the propagation of perturbations (generally called 'defects' in the CA setting) from the previous section to CA, which will shed light on the existing notions of Lyapunov exponents of CA. This involves some theoretical challenges due to essential differences between discrete-time dynamical systems on $\mathbb{R}^n$ and CA acting on $\mathbb{S}^n$. Firstly, the state set of continuous-state dynamical systems is uncountably infinite. For CA, the state set is countable and finite. As such, the classical tools of differential calculus are not available in the

framework of CA. Secondly, the global map governing CA is induced by a local map. This means that defects can spread only to 'neighboring' dimensions in a single time step. For general dynamical systems, there is no notion of 'neighboring' dimensions. First, the concept of path difference needs to be adapted to CA. This can be expressed using modulo 2 addition as follows [3]:

$$\Delta \mathbf{x}^t = \mathbf{f}^t(\mathbf{x}^0) \oplus \mathbf{f}^t(\mathbf{x}^0 \oplus \delta \mathbf{x}^0).$$

$\Delta \mathbf{x}^t$ is generally referred to as the difference pattern of the CA after t step time. Due to the discrete nature of $\mathbb{S}^n$, one cannot introduce an infinitesimally small perturbation vector. Instead, $\delta \mathbf{x}^0$ is a Boolean vector containing ones at the perturbed cells, and zero elsewhere. Typically only a single cell is perturbed, in which case $\delta \mathbf{x}^0$ is simply $\mathbf{e}_i = (0, \ldots, 1, \ldots, 0)$. Unless otherwise specified, the 'difference pattern' refers to the difference pattern emerging from such a simple defect vector. The dynamics of the evolution of the difference pattern can be studied using the CA's jacobian. For this purpose, the notion of a derivative in the framework of CA is required. For a function $f\colon \{0,1\}^n \to \{0,1\}^n$ the Boolean derivative is defined as in [4]:

$$\frac{\partial f_i}{\partial x_j} = f_i(x_1, \ldots, x_j, \ldots, x_n) \oplus f_i(x_1, \ldots, x_j \oplus 1, \ldots, x_n).$$

The Boolean derivative equals 1 if perturbing x_j causes a change in the value of f_i and is 0 otherwise. This notion of derivative for a CA can be used to define the jacobian $J_{\mathbf{f}}(\mathbf{x}^t)$ of a CA at a certain time step. Since $\mathbf{f}$ is induced by a local map ϕ, perturbing a cell x_j can only affect f_i when x_j is in the neighborhood of x_i. Because of this, the jacobian of a CA is tridiagonal. Using the jacobian, the evolution of the difference pattern can be written as [5]:

$$\Delta \mathbf{x}^{t+1} = J_{\mathbf{f}}(\mathbf{x}^t) \otimes \Delta \mathbf{x}^t$$

As one can see from the binary nature of the configuration space (represented in this context by the operation $\otimes$), it is not obvious how to define the "divergence" of the difference path.

It is this difficulty that makes for the definition of a "defect space". By noting that the usual matrix product between the jacobian as previously defined and the difference pattern yields the number of defect paths leading to each lattice cell, one obtains a new space affine to $\mathbb{N}^n$[6]. In this space, the notion of divergence and that of Lyapunov exponent regain their meaning.

2.4 CA Lyapunov spectrum

Following the considerations of the last section, one can now define the Lyapunov exponents as:

$$\lambda_i = \lim_{t \to \infty} \frac{1}{2t} \log(\Lambda_i(t)).$$

$\Lambda_i(t)$ are the square root of the eigenvalue of the matrix $B = MM^T$, and M is obtained by iterating t-times the evolution of the system perturbation through the boolean jacobian matrix:

$$M = J_{\mathbf{f}}(\mathbf{x}^t) \cdots J_{\mathbf{f}}(\mathbf{x}^0).$$

In contrast to discrete dynamical systems, Lyapunov exponents in this context can only take positive values or be equivalent to negative infinity, excluding the possibility of a finite negative value. A negative exponent signifies that an initial perturbation shrinks at an exponential rate. However, this phenomenon is incompatible with cellular automata (CA) because the smallest initial disturbance in this context is a whole integer unit, which can only reduce to zero. At this point, the Lyapunov exponent becomes negative infinity.

3 Numerical implementation

The implementing code is found at https://github.com/samalamadama/Cellular-Automata.
To study cellular automata and their Lyapunov spectra a C++ code was implemented. The code hinges around the "Field" class, characterized by two private variables: rule_ and field_. rule_ is an 8-bit variable (this code makes extensive use of the class boost::dynamic_bitset, which allows for

storing binary variables directly as bits, greatly improving performance) encompassing the evolution rule followed by the automata, defined following the Wolfram convention [7]. In this framework each state of the cell and its neighbors is mapped to a number, naturally taken to be the integer value of their binary representation. So the state "000" will be mapped to 0, "010" to 2 and "110" to 6. The rule associates each of these 8 states with the state of the evolved cell. By ordering each state of the evolved cell in order of the number represented by the whole neighborhood one finds an 8-bit number. For example, the rule indexed by the number "00000000" is the one where for any neighborhood the cell evolves to be in the state "0". For "00001000" the only non-zero evolved state of the cell is the one starting from the neighborhood "4", or "100". The equivalent decimal number is then assigned to the rule, which in this case would be 8. field_ is also a set of binary variables, where the value of the ith element indicates the state of the i-th cell.

The Field class is equipped with the method "evolve" where each cell is evolved following the given rule. The implementing code is:

```
void Field::evolve()
{
  Field const initial_state{*this};
  for (int i = 0; i != (int)field_.size(); ++i){
    field_[i] = rule_[initial_state.get_neighborhood_type(i)];
  }
}

int Field::get_neighborhood_type(int position) const{
  int neighborhood_type;
    if(position==0){
    neighborhood_type = (int)(field_[position+1]<<2|field_[position]<<1|field_[field_.
    size()-1]);
    }else{
      if(position==(int)field_.size()-1){
      neighborhood_type = (int)(field_[0]<<2|field_[position]<<1|field_[position-1]);
      }else{
        neighborhood_type = (int)(field_[position+1]<<2|field_[position]<<1|field_[
    position-1]);
      }
    }
    return neighborhood_type;
}
```

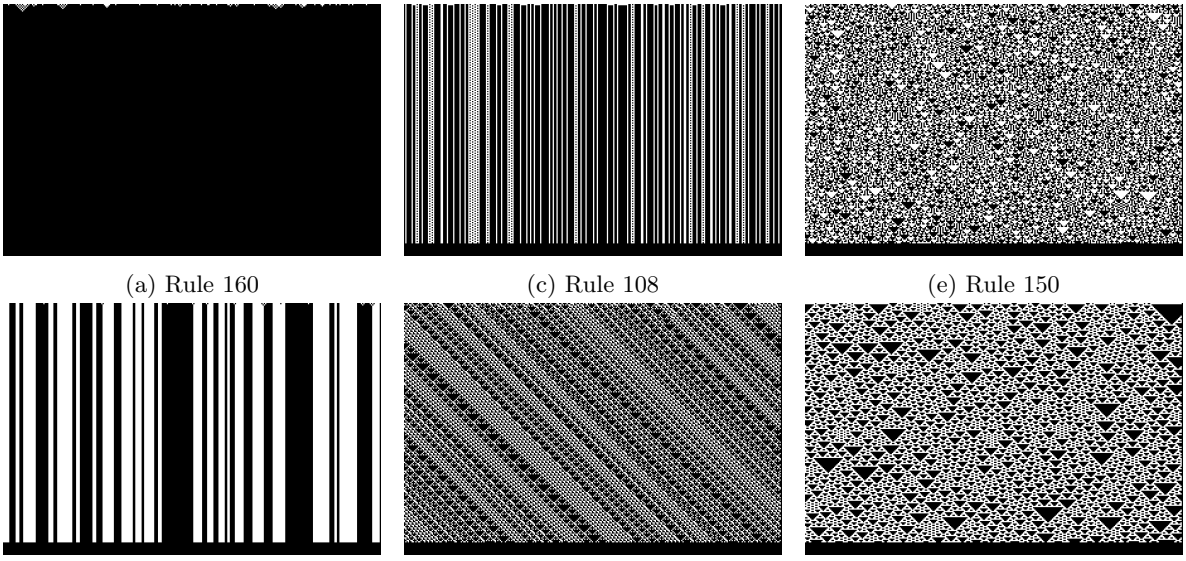
In this section of code, one can see how the type of neighborhood is determined by assigning to it the number previously discussed (the method "$<< n$" moves the bit n significant digits to the left, so $011 << 3$ equals 011000). Special care is taken for the first and last cell of the field, where a periodic boundary condition is implemented (the first cell has the last as its "left" neighbor, while the last has the first as its "right" neighbor). The evolved state of the cell is then taken to be the one assigned by the given rule.

3.1 Damage Spreading graphs

A simple visualization code is then implemented using the SFML library. Following the standard literature, the evolution of an automaton is represented by assigning to each cell a square colored based on its state (black was chosen for "0" and white for "1"). The field is then drawn by placing each of these squares one next to the other. For each evolution the new field is then drawn below the previous, forming a 2D tiling encompassing the dynamics of the field. Some of these evolutions are shown in Figure 1.

The interest of this study lives mainly in how these various rules respond to "damage" in their structure. A defect was thereby introduced at the center of the field (a single cell had its status flipped)and the effects of this defect are shown in Figure 2 (class I automata are omitted as their evolution is of less interest).

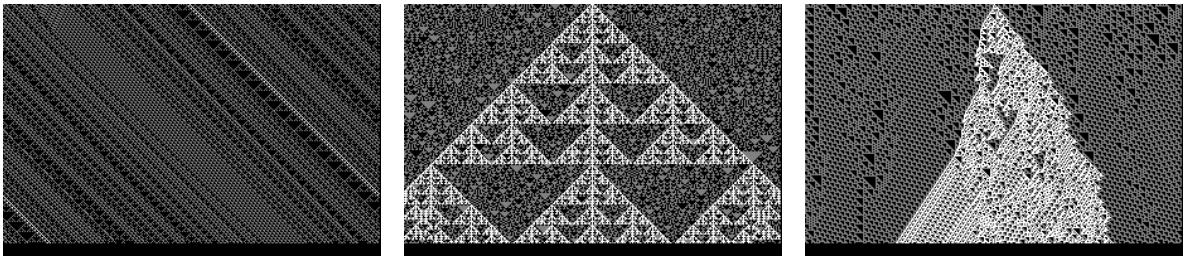
For the numerical analysis, it was made use of the "Eigen" library, which offers an implementation of matrices and methods to solve their spectrum. The matrices used contained "unsigned long long int", giving a maximum value for their coefficients of 2^{64}. The Eigen::Dynamic flag signifies that the dimensions of the matrix are found at runtime. To compute the boolean jacobian as indicated in Section 2.3, the function "jacobian" was introduced. It effectively computes the difference evolution



(a) Rule 160 (c) Rule 108 (e) Rule 150

(b) Rule 232 (d) Rule 26 (f) Rule 126

Figure 1: *Some typical cellular automata patterns. (a) and (b) are class I automata, (c) and (d) are class II, while (e) and (f) are class III*



(a) Rule 26 (b) Rule 150 (c) Rule 110

Figure 2: *The evolution of damages in the cellular automata. The color white represents the cells that are flipped when compared to the unperturbed evolution, the latter represented in black and grey*

pattern of the system under a starting defect in the position indexed by i, this being the column i of the jacobian. The code is:

```
Eigen::Matrix<unsigned long long int, Eigen::Dynamic, Eigen::Dynamic> jacobian(Field
    const& field) {
    int dimension = (int)field.get_field().size();
    Eigen::Matrix<unsigned long long int, Eigen::Dynamic, Eigen::Dynamic> jacob(
    dimension, dimension);
    Field evolved = field;
    evolved.evolve();
    for(int i=0; i!=dimension; ++i){
        Field damaged = field;
        damaged.flip(i);
        damaged.evolve();
        boost::dynamic_bitset<> difference = damaged.get_field() ^ evolved.get_field()
    ;
        for(int j=0; j!=dimension; ++j){
            jacob(j, i) = difference[j];
        }
    }
    return jacob;
}
```

Once one has the discrete jacobian of a field, it becomes possible to compute its Lyapunov spectrum.

The field (500 cells) is made to evolve a set number of times, chosen to be 500 by a posteriori observation of convergence. The jacobian is computed for each evolution step. The product of all the jacobians is then multiplied by its transpose to obtain a symmetric matrix. The spectrum of the resulting matrix is then computed, and the Lyapunov exponents are extracted as previously defined. This is iterated for 40 different initial configurations, and the resulting 20000 coefficient are placed in a vector, to be analyzed. The code is:

```
std::vector<double> lyapunov_spectrum(boost::dynamic_bitset<> rule){
    int n_cells{500};
    int n_evolutions{500};
    int n_iterations{40};
    std::vector<double> spectrum;
    spectrum.reserve(n_cells*n_iterations);

    for(int iteration=0; iteration!=n_iterations; ++iteration){
    Field field{n_cells, -1, rule};
    Eigen::Matrix<unsigned long long int, Eigen::Dynamic, Eigen::Dynamic> Y = Eigen::
    Matrix<unsigned long long int, Eigen::Dynamic, Eigen::Dynamic>::Identity(n_cells,
    n_cells);
    for(int i=0; i!=n_evolutions&&(Y.maxCoeff()<ULLONG_MAX/2); ++i){
        Y = jacobian(field)*Y;
        field.evolve();
    }

    Eigen::MatrixXd L = (Y * Y.transpose()).cast<double>();
    Eigen::SelfAdjointEigenSolver<Eigen::MatrixXd> eigensolver(L);
    if(eigensolver.info() != Eigen::Success) throw std::runtime_error("invalid
    eigensolver");
      for(int i=0; i!= n_cells; ++i){
      spectrum.push_back(std::log(eigensolver.eigenvalues()(i))/(2*n_evolutions));
      }

      std::cout<<(iteration+1)*100/n_iterations<<"%\n";
    }
    return spectrum;
}
```

As it is clear from our definition of Lyapunov exponents, the eigenvalues grow exponentially with time. This poses the biggest challenge of this approach, as for high values of t (number of iterations) the eigenvalues (and in general the elements of the matrix) exceed the number of digits allowed with the standard C++ types. For the worst case rule, rule 150, which has a jacobian independent of the specific starting configuration and equal to:

$$J = \begin{bmatrix} 1 & 1 & 0 & \dots & 0 & 1 \\ 1 & 1 & 1 & \dots & 0 & 0 \\ & \vdots & & \ddots & \vdots & \\ 1 & 0 & 0 & \dots & 1 & 1 \end{bmatrix}$$

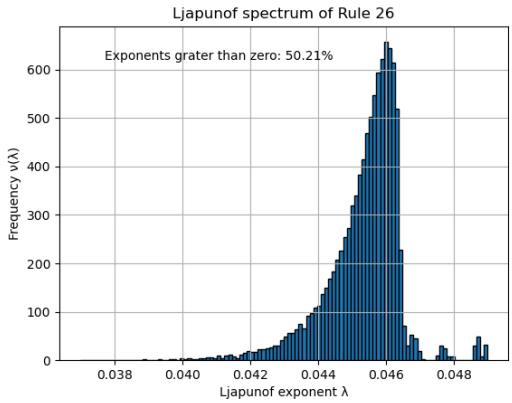
Its corresponding maximum Lyapunov coefficient is found to be 1. Reversing the earlier formula one may see that the corresponding eigenvalue (after 500 iterations) is of the order of e^{500}, which has $\simeq 722$ binary digits!

The computing time required to diagonalize such matrices also grows fast with the size of their elements. This constrained our analysis to rule which are not divergent, e.g. class I and II. Class I is trivial, all Lyapunov exponents being equivalent to $-\infty$. Class II is instead not trivial, showing localized chaoticity which quickly decays in stable patterns, which will be studied in the next section. Their inherent stability keeps the matrix coefficients from growing too quickly (note how in the code the iteration is stopped if one of the coefficients of the matrix exceeds the binary equivalent of 63 bits). This allows for their spectrum to be explicitly calculated. On our setup, this required $\simeq 2$ h per rule.

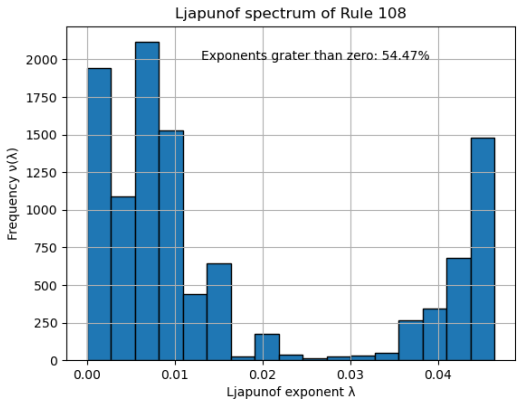
3.2 Lyapunov spectra graphs

In this section, the Lyapunov spectra obtained through the procedure described earlier are presented. CLASS I: After a transient, class I rules settle on a homogeneous configuration. As such, damage cannot spread and the Lyapunov spectrum comprises exclusively of $\lambda = -\infty$ exponents. Such exponents are

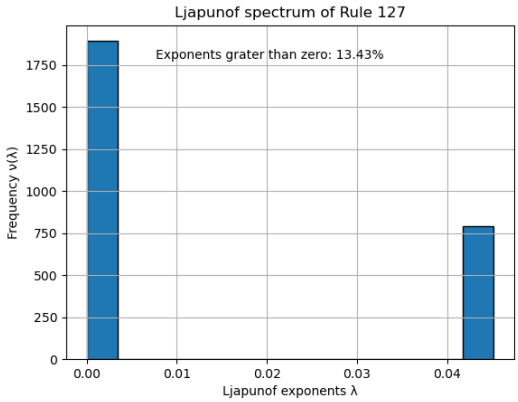
called superstable.
CLASS II: Some rules in class II yield relatively simple behavior, corresponding to constant space-time patterns or periodic patterns with a temporal period of 2 [3]. Some of these rules yield a spectrum of superstable exponents, similarly to class I rules, indicating that any initial defect will always die out. Conversely, some simple rules yield a spectrum that is a combination of superstable and stable (i.e. $\lambda = 0$) exponents. This indicates that some defects will die out, while others will be propagated to a later time step, yet the difference pattern does not grow and its size remains constant and equal to 1. Most class II rules with a period larger than 2 yield a spectrum that is a combination of superstable and unstable (i.e. $\lambda \geq 0$) exponents. This indicates that some defects will die out, while others will yield a difference pattern that grows to a fixed size [3]. Such class II rules exhibit chaotic behavior within specific boundaries, leading some authors to refer to them as locally chaotic rules [8].



(a) Rule 26



(b) Rule 108



(c) Rule 127

Figure 3: Lyapunov spectra of class II rule 26 (a), 108 (b), and 127 (c), The x-axis denotes the Lyapunov exponent values, and the y-axis indicates the frequency of occurrence for each value in the spectrum. At the top of the spectra, the percentage of exponents that are not $-\infty$ is shown.

4 Conclusion

The Lyapunov spectra found for the examined rules behave as expected. Even though the connection between Lyapunov exponents as earlier defined and the evolution in the configuration space of the system is not straightforward, one expects class II automata to have a high fraction of its Lyapunov exponents to be $-\infty$, indicating that a random mutation of a cell is likely to die out quickly. This is found to be the case. Analyzing closer the spectrum of rule 108, it can be seen how it presents a

peak around the value 0. This can be interpreted by observing the characteristic "streams" of rule 108, which once created remain present for every successive time t. It can be seen then how a random mutation might not die out, but also wouldn't spread, remaining localized. This is equivalent to a 0 Lyapunov exponent. Lastly, the analyzed rules present a narrow peak around some (low) values of their spectra, indicating a local chaoticity. This is to be compared to the spectra of class III and IV, which present less (or none) stable exponents and a wider distribution centered around higher values of Lyapunov exponents [5]

References

[1] Stephen Wolfram. Universality and complexity in cellular automata. *Physica D: Nonlinear Phenomena*, 10(1-2):1–35, 1984.

[2] James J Binney, Nigel J Dowrick, Anthony J Fisher, and Mark EJ Newman. *The theory of critical phenomena: an introduction to the renormalization group*. Oxford University Press, 1992.

[3] Milan Vispoel, Aisling J Daly, and Jan M Baetens. Damage spreading and the lyapunov spectrum of cellular automata and boolean networks. *Chaos, Solitons & Fractals*, 184:114989, 2024.

[4] Gérard Y Vichniac. Boolean derivatives on cellular automata. *Physica D: Nonlinear Phenomena*, 45(1-3):63–74, 1990.

[5] Baetens J. M. Vispoel M., Daly A. J. Damage spreading and the lyapunov spectrum of cellular automata and boolean networks. *Chaos, Solitons & Fractals, 184, 114989*, 2024.

[6] Franco Bagnoli, R Rechtman, and Stefano Ruffo. Damage spreading and lyapunov exponents in cellular automata. *Physics Letters A*, 172(1-2):34–38, 1992.

[7] S Wolfram. A new kind of science (wolfram media, inc., champaign il, usa). 2002.

[8] Wentian Li, Norman Packard, et al. The structure of the elementary cellular automata rule space. *Complex systems*, 4(3):281–297, 1990.