TESI DOCTORAL

Títol Aspects of algorithms and dynamics of cellular paradigms

Realitzada per Giovanni E. Pazienza

en el Centre Enginyeria i Arquitectura La Salle

i en el Departament Electrónica y Comunicaciones

Dirigida per Xavier Vilasís Cardona

Aspects of algorithms and dynamics of cellular paradigms

Giovanni E. Pazienza

Ph.D. dissertation

Supervisor: Dr. Xavier Vilasís-Cardona

Universitat Ramon Llull, December 2008

to Pedro, who is fighting for life

Contents

Chapter 1

Introduction

In the last few years, we have witnessed a revolution in the field of computing architectures, since the multi-core paradigm has been extensively applied to commercial processors that dominate the market of home computers and gaming consoles. Modern graphics cards, which integrate dozens of processing units implementing a real parallel hardware, are a clear example of this tendency [1]. A similar approach has found room in many fields, including arrays of tactile sensors [2] and nanoantennas [3]. Last but not least, this concept has a physiological counterpart, as recent studies on the human brain processes show [4].

Moreover, the technical advances in electronic device manufacturing have encouraged architectures based on simple processing cores locally connected. This arrangement shortens the buses, resulting in two positive effects: firstly, it reduces the amount of wiring required hence favoring integration; secondly, it allows to overcome the problem of transmitting a high frequency clock signal uniformly throughout the whole chip. Furthermore, the possibility of integrating sensors on chip [5] has opened new scenarios in which this novel approach plays a key role. In 2007 Cellular Neural Network devices were indicated as one of the most promising among the emerging research materials by the International Technology Roadmap for Semiconductors in its annual report [6].

In the following, we will refer to arrays of processing units arranged in ordered structures (usually two-dimensional grids) as *cellular paradigms*. This notion allows us to abstract our conclusions from a specific problem, and then they will be valid whichever the field of application. The concept of cellular paradigm involves three different levels of characterization: devices, architectures, and algorithms.

As for the first level, we already mentioned how successfully the cellular paradigm has been employed in cutting-edge general-purpose processors, like

IBM Blue Gene supercomputer [7], IBM-Sony-Toshiba CELL multiprocessor chip [8], and Intel 80-tile processor chip [9].

The second point, regarding the definition of a suitable architecture for this new generation of devices, is a big challenge. This issue is indeed anterior to the creation of parallel hardware, and both Turing and von Neumann conjectured computational structures taking advantage of the synergy among processing elements locally connected. Their studies were the precursors of the modern cellular architectures, including the Cellular Neural Network (CNN), introduced in 1988 by Chua and Yang. On the ground of the studies on CNNs, Roska and Chua defined the 'Cellular Neural Network - Universal Machine' (CNN-UM), a supercomputer combining analog and digital operations into a unique computing structure. The natural extension of this architecture is the 'Cellular Wave Computer' (CWC) , a machine working on flows — e.g., video streams — that represents a breakthrough in research on cellular paradigms: it goes beyond the Boolean logic, working on analogic inputs and using partial differential equations as elementary instructions.

While the issue of defining devices and architectures for cellular paradigms has already been addressed, the third aspect - algorithms - is far from being completely understood. In fact, there are still numerous open questions on this subject, some of which will be answered in this dissertation. In our analysis, we follow two main threads, both regarding theoretical and practical aspects of processing on cellular paradigms.

The first thread concerns programs for the Cellular Wave Computer which, being universal, is capable of performing any computable algorithm, if properly programmed. In our search for a systematic way to design programs for the CWC, we focused on the parallelism between the CNN-UM and another computation paradigm called 'Genetic Programming + Indexed Memory machine'. This approach allowed us to find an alternative proof of universality for the CNN-UM as well as two other remarkable results: firstly, a general form for all CNN-UM programs; secondly, the feasibility of the exploration of the solution space by means of a machine learning technique called 'Genetic Programming' (GP). As confirmed by the experiments reported here, the method we propose results to be a valid aid to create effective algorithms to deal with all sort of image processing tasks.

The second thread regards 'Cellular Automata' (CA), a particular case of Cellular Wave Computer already proposed by von Neumann, but popularized by Stephen Wolfram through his book "A new kind of science" which settles the bases for a systematic study of this topic. Recently, Leon Chua has analyzed CA through the means of nonlinear dynamics in his series of books entitled "A nonlinear dynamics perspective of Wolfram's new kind of science", providing rigorous explanations for some well-known phenomena and presenting numerous new results. Our contribution has embraced a number of aspects of CA, such as the emergence of fractals and the study of the so-called *Bernoulli σ_τ-attractors.*

The dissertation reflects the structure illustrated so far.
Chapter 2 reviews cellular paradigms, introducing the nomenclature and sketching their physical implementations with a particular focus on simulation on graphics cards, one of the most promising applications for the future.
Chapter 3 starts with fundamental notions of theory of computation, which are then used to explain the significance of the proof of universality of the CNN-UM detailed further in the same chapter.
Chapter 4 includes an overview of Genetic Programming as well as the description of its application in the automatic design of programs for the CNN-UM.
Chapter 5 focuses on the experiments in which our approach has proved to be effective, illustrating also what kinds of practical problems the designer faces when dealing with real-life applications.
Chapter 6 includes general explanations of the nonlinear dynamics perspective of Cellular Automata, describing the main characteristics of this approach and introducing the basic tools for the analysis.
Chapter 7 is completely dedicated to new results found through the nonlinear dynamics perspective of Cellular Automata; the work on this topic is still in progress, hence we have chosen to include here only a few results regarding particular aspects.
Finally, Chapter 8 draws the conclusions of the work and suggests possible scenarios for the future.

Chapter 2

Cellular paradigms

The last findings of the electronic industry suggest that one way to keep on integrating technology though controlling the power dissipation is relying on simple processing units locally connected. Thanks to this approach, usually referred to as ‘cellular’, nowadays it is possible to manufacture devices with a billion of transistors at nanometer scale size integrating sensing and processing, which are the basis for high-speed processing cameras.

Physiologists point out that human brain differs from a modern computer in many aspects: for instance, signals are not digital but analog, there is no centralized clock, and spatial arrangement of processing units matters. These considerations have led to the search of computation models capable of mimicking the brain functionalities: neural networks and fuzzy logic are probably two of the most-known examples of such attempts. Nevertheless, the application of such studies have often remained limited to particular spheres of interest, and they failed to become a real alternative to the von Neumann architecture.

Many elements indicate that cellular paradigms can be the link between computer science, since they are an efficient and effective computation model, and physiology, because they find impressive analogies in the way our brain works.

In this chapter we go through some of the cellular paradigms that have been considered in this thesis, exploring the relationship among them, and presenting a brief overview of their electronic implementations.

2.1 Cellular Neural Networks

Cellular Neural Networks (CNNs) [10] [11] are arrays of dynamical artificial systems (cells) with local connections only. Cells can be arranged in several configurations (e.g. [12] [13]), however here we consider only two-dimensional CNNs organized in a eight-neighbor rectangular grid; this choice is justified by the fact that other CNNs of various regular grids can be mapped onto this configuration by applying weight matrices of periodic space variance [14]. A visual representation of a typical 2D Cellular Neural Network is shown in Fig. 2.1.

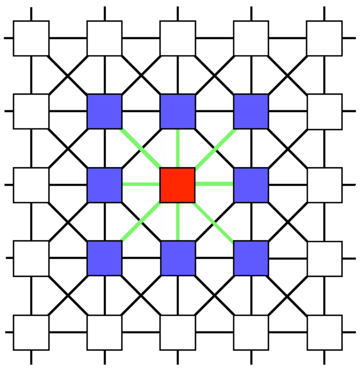


Figure 2.1: Two-dimensional CNN: the red cell has nine neighbors (the eight blue cells and itself) to which is connected through the green connections.

In the standard model, the output of each cell depends only on the input and the output of its neighbor cells; the number of neighbors is usually defined by the *radius of neighborhood* r: when $r = 1$, the neighborhood includes the cell itself plus its eight nearest cells; when $r = 1$ we add to this set 16 more cells, corresponding to a further level of surrounding cells. A CNN cell can be thought as an electrical circuit, whose dynamics can be analyzed through the usual mathematical tools of systems theory. In particular, each cell of the so-called Chua-Yang model contains linear (resistors, capacitors, controlled sources) and nonlinear (controlled sources) circuit elements, as well as independent sources (see Fig. 2.2). Since variables are continuous valued signals, this model is also called continuous-time CNN.
For the sake of clarity, we denote by u_{ij} the input voltage of the cell in position (i, j); analogously, x_{ij} and y_{ij} are the state and the output voltages, respectively, of the same cell. As for the controlled current sources, their equations are

$$I_{xu} = B(i, j; k, l) \cdot u_{kl}; \quad I_{xy} = A(i, j; k, l) \cdot y_{kl}; \quad I_{yx} = \frac{1}{R_y} f(x_{ij}) \tag{2.1}$$

where the indices k and l denote a generic cell belonging to the neighborhood of the cell in position (i, j). Therefore, the dynamics of the array can be

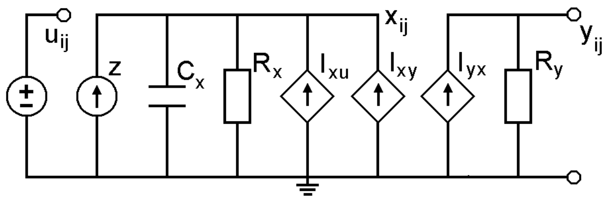


Figure 2.2: A standard CNN cell.

described by the state equation

$$\begin{aligned} C\frac{dx_{ij}(t)}{dt} &= -\frac{1}{R_x}x_{ij}(t) + \sum_{(k,l)\in\mathcal{N}(i,j)} A(i,j;k,l)\cdot y_{kl}(t) \\ &+ \sum_{(k,l)\in\mathcal{N}(i,j)} B(i,j;k,l)\cdot u_{kl}(t) + z(i,j;k,l) \end{aligned} \tag{2.2}$$

In this notation $\mathcal{N}(i,j)$ is the set of indexes corresponding to cell (i,j) itself and its neighborhood. For example, a common assumption in CNNs is using a radius 1 neighborhood, which means that the 9 neighbors of the cell (i,j) have indices (k,l) with $1\leq k\leq 3$ and $1\leq l\leq 3$. Note that often the value R_xC is interpreted as a time constant τ.
The set of matrices $\{A,B,z\}$, usually called *cloning template*, defines the operation performed by the network, and, in general, the values of A, B, and z vary for each pair of cells (i,j) and (k,l). However, the case in which the weights of the network depend on the difference between cell indices is particularly interesting to analyze, because few parameters - the values of the cloning template - set the behavior of the whole network. This kind of CNN is called ‘space-invariant’, and it will be used throughout this work. For instance, when radius 1 neighborhood is used, A and B are 3×3 matrices, z is a scalar, and all values are independent of the indices (i,j,k,l):

$$A=\begin{pmatrix} a_{-1,-1} & a_{-1,0} & a_{-1,1}\\ a_{0,-1} & a_{0,0} & a_{0,1}\\ a_{1,-1} & a_{1,0} & a_{1,1}\end{pmatrix} \quad B=\begin{pmatrix} b_{-1,-1} & b_{-1,0} & b_{-1,1}\\ b_{0,-1} & b_{0,0} & b_{0,1}\\ b_{1,-1} & b_{1,0} & b_{1,1}\end{pmatrix} \quad z=z_{0,0}$$

The expression for the output voltage y_{ij} can be easily obtained by Fig. 2.2 and Eq. 2.1. Usually, it is assumed that $R_y=1$ and $f(x_{ij}(t))$ is as follows

$$\begin{aligned} y_{ij}(t) &= f(x_{ij}(t)) = \tfrac{1}{2}(|x_{ij}(t)+1| -| x_{ij}(t)-1|) = \\ &= \begin{cases} 1, & \text{if } x_{ij}\geq 1\\ x_{ij}, & \text{if } -1\leq x_{ij}\leq 1\\ -1, & \text{if } 1\leq x_{ij}\end{cases} \end{aligned} \tag{2.3}$$

even though other functions $f(\cdot)$ are also possible [15].
It is not difficult to prove that all states x_{ij} in a CNN are bounded for any $t > 0$ (see Theorem 1 in [10]) and the convergence of the network can be studied through the analysis of the Lyapunov function. Some simple conditions on the circuit parameters guarantee the stability of the network, in particular the CNN can be set to have only binary (± 1) output y_{ij}. Given a *cloning template*, the steady state reached by the network depends on the initial state and the input. There exist alternative CNN models with better characteristics in particular contexts, like the 'discrete-time' CNN (DT-CNN) [16], and the 'full signal range' CNN (FSR-CNN) [17].
Image processing is one of the main applications of CNNs: images are fed into the network as initial state and/or input, associating each cell of the CNN with a pixel of the image, whereas the matrices A, B and z define the operation to perform; the network settings to execute a great number of tasks can be found in [18]. A single-layer CNN has some computational limitations, however it is possible to prove that a multi-layer CNN is as universal as a Turing machine, and then it can deal with *any* computable algorithm [19], as detailed in chapter 3.

2.2 CNN Universal Machine

The potentialities of a Cellular Neural Network can be fully exploited by using it as computing core of a paradigm called CNN-Universal Machine (CNN-UM) [20] whose architecture, composed by a grid of CNN nuclei with extended capabilities controlled by a *Global Analogic Programming Unit* (GAPU), is shown in Fig. 2.3 [21].
The GAPU consists of four main functional blocks: first, the *Global Analogic Control Unit* (GACU) that stores the CNN-UM program and synchronizes the communication with external controlling devices; second, the *Analog Program Register* (APR) that contains the CNN templates used as instructions of the program in the GACU; third, the *Logic Program Register* (LPR) that includes the control sequences for the individual cells LLU; fourth, the *Switch Configuration Register* (SCR) that stores the switch states governing the cell configurations used in the CNN-UM program.
The elements extending the standard CNN nucleus are: two memories, one for for analog values - *Local Analog Memory* (LAM) - and one for logic values - *Local Logical Memory* (LLM) - used to store variables in each cell; a *Local Analog Output Unit* (LAOU) and a *Local Logic Unit* (LLU) to execute cell-wise analog and logic operations, respectively, on the storable values; and finally, a *Local Communication and Control Unit* (LCCU) that is in charge

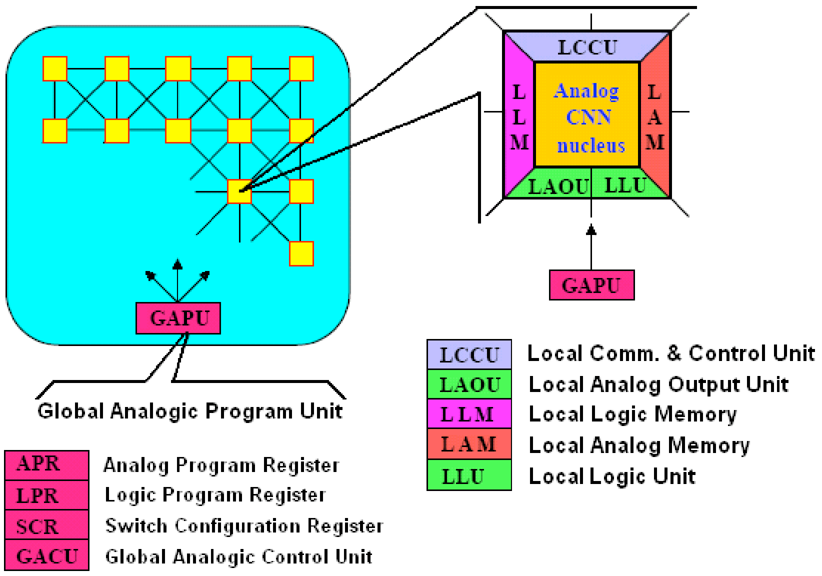

LCCU
Analog CNN nucleus
L L M
L A M
LAOU
LLU
GAPU
GAPU
Global Analogic Program Unit
APR Analog Program Register
LPR Logic Program Register
SCR Switch Configuration Register
GACU Global Analogic Control Unit
LCCU Local Comm. & Control Unit
LAOU Local Analog Output Unit
LLM Local Logic Memory
LAM Local Analog Memory
LLU Local Logic Unit

Figure 2.3: Scheme of the CNN-UM.

of communicating with the GAPU, according to the mechanism described in [22].
A typical CNN-UM program is composed by a sequence of logic and analog operations, defined by either linear or nonlinear templates. Intermediate results are storable in the local memories (LAMs and LLMs), whereas the final output can be defined both in fixed and non-fixed state of the network (equilibrium and non-equilibrium computing) depending on the control of the transient length.

2.3 Cellular wave computer

A Cellular Wave Computer [23] is an extension of the Cellular Neural Network - Universal Machine in which input and output are image flows and elementary instructions are differential equations. The philosophy of this paradigm is completely different from the one of a standard digital computer: in some sense it is the practical realization of what von Neumann called *analytic theory of computing* [24] as well as the basis for generating the Turing patterns [25]. Problems intractable with 'traditional' computers, like a typical reaction-diffusion equation, become trivial in this architecture because they are performed by a single instruction.
In the case of 2D image flows the only discretization is in space, and a finite

time image flow $\Phi(t)$ is defined as

$$\Phi(t) := \{\varphi_{ij}(t), t \in [0, t_d]\},$$

where $1 \leq i \leq m$ and $1 \leq j \leq n$ (considering image flows with size $m \times n$), $t_d > 0$ (time duration), $\varphi_{ij} \in C^1$ (continuously differentiable) and bounded.
At $t = t^*$, $\mathcal{P} = \Phi(t^*)$ is a $m \times n$ picture

$$\mathcal{P} : \ p_{ij} \in \mathbb{R}$$

A binary picture is usually called *mask* $\mathcal{M}$, hence

$$\mathcal{M} : \ m_{ij} \in \{-1, 1\} \ \text{(or equivalently, } m_{ij} \in \{0, 1\})$$

The elementary instruction, also called *wave instruction*, of the Cellular Wave computer operating on the input flow $\Phi_{input}(t)$ is defined as

$$\Phi_{output}(t) := \Psi(\Phi_{input}(t), \mathcal{P}, \partial);$$

being Ψ a function on image flows, $\mathcal{P}$ a picture defining initial state and/or bias map, and ∂ the boundary conditions.
A matrix functional Γ is defined as

$$\mathcal{P}_{output} := \Gamma(\Phi_{input}(t), \mathcal{P}, \partial);$$

Both Ψ and Γ operate on image flows (i.e. a video stream), but in the first case the output is another image flow $\Phi_{output}(t)$ whereas in the the other the output is a picture $\mathcal{P}_{output}$.
Finally, a scalar functional γ is defined as

$$q := \gamma(\Phi_{input}(t), \mathcal{P}, \partial);$$

where q is a real number. Examples of scalar functionals are the *Global White* operator, detecting the presence of at least one black pixel in the flow, and the *Nonlinear Hausdorff wave metric*, measuring closeness of an object in Φ_{input} to $\mathcal{P}$ (see Sec. 4.4.3).
Not always the output is settled for $t > t_d$, but there may exist grid positions (i,j) for which $\frac{d\varphi_{ij}}{dt} \neq 0$; in this case, the spatial-temporal instruction is called of *non-equilibrium type*.

The spatial-temporal algorithms are called analogic because they are analog and logic at the same time, and they are described through the so-called *α-recursive functions*, defined by

- initial settings of image flows, pictures, masks, and boundary values: $\Phi(0)$, $\mathcal{P}$, $\mathcal{M}$, ∂, respectively;

- equilibrium and non-equilibrium solutions of partial differential difference equations defined via the canonical CNN equations on $\Phi(t)$;
- global (and local) minimization on the above;
- arithmetic and logic combinations of the results of the above operations, analog comparisons (thresholding), and logic conditions;
- recursions on the above operations.

A collection of analogic algorithms can be found in [18], whereas the symbols describing them will be analyzed in Sec. 3.4.1.

2.4 Cellular automata

Cellular automata (CA) consist of regular uniform lattice of cells assuming a finite number of states; here, we consider one-dimensional CA in which cells are arranged in an array of length $L = I + 1$ and can take only two states, 0 and 1. For instance, a generic bit string $\mathbf{x}$ is

$$\mathbf{x} = (x_0 x_1 \ldots x_{I-1} x_I)$$

where the subscript indicates the position of the cell in the array.
Cells are updated synchronously (discrete-time evolution) and, introducing a superscript indicating the iteration, it is possible to summarize the time evolution of a bit string as

$$\mathbf{x}_i^{n+1} = f(\mathbf{x}^n) \tag{2.4}$$

The state of each cell at iteration $n + 1$ depends on the states of the cells in its neighborhood (here we consider only the nearest neighbors) at iteration n

$$x_i^{n+1} = f(x_{i-1}^n x_i^n x_{i+1}^n) \tag{2.5}$$

and this dependence can be conveniently represented through a truth table, as shown in Table 2.1.
The values β_k are binary, $\beta_k \in \{0, 1\}$, and the array $(\beta_8 \beta_7 \ldots \beta_0)$ defines univocally the behavior of the Cellular Automaton. Therefore, there are $2^{2^3} = 256$ ways to set the function f in Eq. 2.5, corresponding to all outcomes of the array $(\beta_8 \beta_7 \ldots \beta_0)$ in Table 2.1. Each of these functions is called *local rule* $\mathcal{N}$, and the univocal correspondence between a rule and a truth table is defined through the formula

$$\mathcal{N} = \sum_{i=0}^{7} \beta_i \cdot 2^i$$

Table 2.1: Truth table for a CA local rule.

$x_{i-1}^n x_i^n\ x_{i+1}^n$	x_i^{n+1}
000	β_0
001	β_1
010	β_2
011	β_3
100	β_4
101	β_5
110	β_6
111	β_7

hence, rules are numbered from 0 to 255.
Therefore, we denote the transformation of the bit string $\mathbf{x^n}$ into $\mathbf{x^{n+1}}$ under rule $\mathcal{N}$ by

$$\mathbf{x^{n+1}} = T_{\mathcal{N}}(\mathbf{x^n})$$

and the evolution of a generic cell x_i^n by

$$x_i^{n+1} = T_{\mathcal{N}}(x_{i-1}^n x_i^n x_{i+1}^n)$$

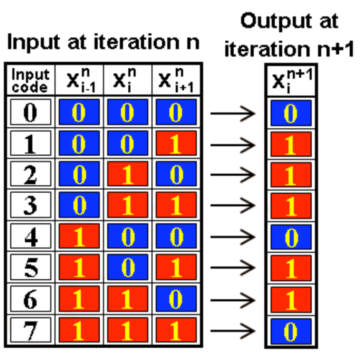
In the following, we use periodic boundary conditions, which means

$$x_0^{n+1} = T_{\mathcal{N}}(x_I^n x_0^n x_1^n) \quad \text{and} \quad x_I^{n+1} = T_{\mathcal{N}}(x_{I-1}^n x_I^n x_0^n)$$

Equivalently, rules can be described through *Boolean cubes*, in which the table truth is mapped onto the vertices of a cube, like in the example of Fig. 2.4.
Cellular Automata were originally introduced by Ulam [26] and von Neumann [27] with the purpose of modeling biological system self-reproduction; however, their divulgation is mainly due to Wolfram [28], who defined the notation, pointed out interesting properties, and proposed possible applications. In the last years, the analysis of CA from a 'nonlinear dynamics perspective' [29] has provided new elements to this discipline. For example, it has been possible to prove that the 256 rules in fact belong to only 88 equivalence classes: in particular, each rule $\mathcal{N}$ can be transformed into an equivalent one by using one of the three global equivalence transformations - *left-right transformation* $T^{\dagger}(\mathcal{N})$, *global complementation* $\overline{T}(\mathcal{N})$, and *left-right complementation* $T^{*}(\mathcal{N})$ - whose details are explained in [30]. The 256 local rules and their global equivalent rules are shown in Table 2.2.

Table 2.2: CA local rule $\mathcal{N}$ (first column), left-right transformation $T^\dagger(\mathcal{N})$ (second column), global complementation $\overline{T}(\mathcal{N})$ (third column), left-right complementation $T^*(\mathcal{N})$ (fourth column).

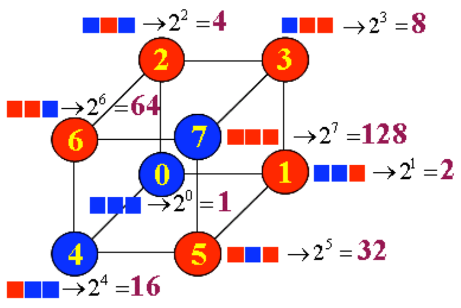
$\mathcal{N}$	$T^\dagger(N)$	$\overline{T}(\mathcal{N})$	$T^*(\mathcal{N})$
0	0	255	255
1	1	127	127
2	16	191	247
3	17	63	119
4	4	223	223
5	5	95	95
6	20	159	215
7	21	31	87
8	64	239	253
9	65	111	125
10	80	175	245
11	81	47	117
12	68	207	221
13	69	79	93
14	84	143	213
15	85	15	85
16	2	247	191
17	3	119	63
18	18	183	183
19	19	55	55
20	6	215	159
21	7	87	31
22	22	151	151
23	23	23	23
24	66	231	189
25	67	103	61
26	82	167	181
27	83	39	53
28	70	199	157
29	71	71	29
30	86	135	149
31	87	7	21
32	32	251	251
33	33	123	123
34	48	187	243
35	49	59	115
36	36	219	219
37	37	91	91
38	52	155	211
39	53	27	83
40	96	235	249
41	97	107	121
42	112	171	241
43	113	43	113
44	100	203	217
45	101	75	89
46	116	139	209
47	117	11	81
48	34	243	187
49	35	115	59
50	50	179	179
51	51	51	51
52	38	211	155
53	39	83	27
54	54	147	147
55	55	19	19
56	98	227	185
57	99	99	57
58	114	163	177
59	115	35	49
60	102	195	153
61	103	67	25
62	118	131	145
63	119	3	17



Input code	x^n_{i-1}	x^n_i	x^n_{i+1}		x^{n+1}_i
0	0	0	0	$\longrightarrow$	0
1	0	0	1	$\longrightarrow$	1
2	0	1	0	$\longrightarrow$	1
3	0	1	1	$\longrightarrow$	1
4	1	0	0	$\longrightarrow$	0
5	1	0	1	$\longrightarrow$	1
6	1	1	0	$\longrightarrow$	1
7	1	1	1	$\longrightarrow$	0

Input at iteration n
Output at iteration n+1

(a)



(b)

Figure 2.4: (a) Truth table and (b) Boolean cube for rule 110. Note the correspondence between input codes of the truth table and vertices of the Boolean cube.

$\mathcal{N}$	$T^\dagger(N)$	$\overline{T}(\mathcal{N})$	$T^*(\mathcal{N})$
64	8	253	239
65	9	125	111
66	24	189	231
67	25	61	103
68	12	221	207
69	13	93	79
70	28	157	199
71	29	29	71
72	72	237	237
73	73	109	109
74	88	173	229
75	89	45	101
76	76	205	205
77	77	77	77
78	92	141	197
79	93	13	69
80	10	245	175
81	11	117	47
82	26	181	167
83	27	53	39
84	14	213	143
85	15	85	15
86	30	149	135
87	31	21	7
88	74	229	173
89	75	101	45
90	90	165	165
91	91	37	37
92	78	197	141
93	79	69	13
94	94	133	133
95	95	5	5
96	40	249	235
97	41	121	107
98	56	185	227
99	57	57	99
100	44	217	203
101	45	89	75
102	60	153	195
103	61	25	67
104	104	233	233
105	105	105	105
106	120	169	225
107	121	41	97
108	108	201	201
109	109	73	73
110	124	137	193
111	125	9	65
112	42	241	171
113	43	113	43
114	58	177	163
115	59	49	35
116	46	209	139
117	47	81	11
118	62	145	131
119	63	17	3
120	106	225	169
121	107	97	41
122	122	161	161
123	123	33	33
124	110	193	137
125	111	65	9
126	126	129	129
127	127	1	1

$\mathcal{N}$	$T^\dagger(N)$	$\overline{T}(\mathcal{N})$	$T^*(\mathcal{N})$
128	128	254	254
129	129	126	126
130	144	190	246
131	145	62	118
132	132	222	222
133	133	94	94
134	148	158	214
135	149	30	86
136	192	238	252
137	193	110	124
138	208	174	244
139	209	46	116
140	196	206	220
141	197	78	92
142	212	142	212
143	213	14	84
144	130	246	190
145	131	118	62
146	146	182	182
147	147	54	54
148	134	214	158
149	135	86	30
150	150	150	150
151	151	22	22
152	194	230	188
153	195	102	60
154	210	166	180
155	211	38	52
156	198	198	156
157	199	70	28
158	214	134	148
159	215	6	20

$\mathcal{N}$	$T^\dagger(N)$	$\overline{T}(\mathcal{N})$	$T^*(\mathcal{N})$
160	160	250	250
161	161	122	122
162	176	186	242
163	177	58	114
164	164	218	218
165	165	90	90
166	180	154	210
167	181	26	82
168	224	234	248
169	225	106	120
170	240	170	240
171	241	42	112
172	228	202	216
173	229	74	88
174	244	138	208
175	245	10	80
176	162	242	186
177	163	114	58
178	178	178	178
179	179	50	50
180	166	210	154
181	167	82	26
182	182	146	146
183	183	18	18
184	226	226	184
185	227	98	56
186	242	162	176
187	243	34	48
188	230	194	152
189	231	66	24
190	246	130	144
191	247	2	16

$\mathcal{N}$	$T^\dagger(N)$	$\overline{T}(\mathcal{N})$	$T^*(\mathcal{N})$
192	136	252	238
193	137	124	110
194	152	188	230
195	153	60	102
196	140	220	206
197	141	92	78
198	156	156	198
199	157	28	70
200	200	236	236
201	201	108	108
202	216	172	228
203	217	44	100
204	204	204	204
205	205	76	76
206	220	140	196
207	221	12	68
208	138	244	174
209	139	116	46
210	154	180	166
211	155	52	38
212	142	212	142
213	143	84	14
214	158	148	134
215	159	20	6
216	202	228	172
217	203	100	44
218	218	164	164
219	219	36	36
220	206	196	140
221	207	68	12
222	222	132	132
223	223	4	4

$\mathcal{N}$	$T^\dagger(N)$	$\overline{T}(\mathcal{N})$	$T^*(\mathcal{N})$
224	168	248	234
225	169	120	106
226	184	184	226
227	185	56	98
228	172	216	202
229	173	88	74
230	188	152	194
231	189	24	66
232	232	232	232
233	233	104	104
234	248	168	224
235	249	40	96
236	236	200	200
237	237	72	72
238	252	136	192
239	253	8	64
240	170	240	170
241	171	112	42
242	186	176	162
243	187	48	34
244	174	208	138
245	175	80	10
246	190	144	130
247	191	16	2
248	234	224	168
249	235	96	40
250	250	160	160
251	251	32	32
252	238	192	136
253	239	64	8
254	254	128	128
255	255	0	0

Despite the limited number of global independent rules, summarized in Table 2.3, CA can show an extraordinary variety of behaviors: some rules have no practical interest (like 0 or 255, mapping the whole space onto a single point), others are even capable of universal computation, like rule 110.

A further intrinsic characteristic of a local rule is its *complexity index* κ, defined as the number of planes needed to separate blue and red vertices - corresponding to the two states (0 and 1, respectively) - in its Boolean cube.

Table 2.3: 88 global equivalence classes for CA local rules.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
8	**9**	**10**	**11**	**12**	**13**	**14**	**15**
18	**19**	**22**	**23**	**24**	**25**	**26**	**27**
28	**29**	**30**	**32**	**33**	**34**	**35**	**36**
37	**38**	**40**	**41**	**42**	**43**	**44**	**45**
46	**50**	**51**	**54**	**56**	**57**	**58**	**60**
62	**72**	**73**	**74**	**76**	**77**	**78**	**90**
94	**104**	**105**	**106**	**108**	**110**	**122**	**126**
128	**130**	**132**	**134**	**136**	**138**	**140**	**142**
146	**150**	**152**	**154**	**156**	**160**	**162**	**164**
168	**170**	**172**	**178**	**184**	**200**	**204**	**232**

In general, $\kappa \in \{1, 2, 3\}$: rules with $\kappa = 1$ are in red in Tables 2.2 and 2.3, in blue when $\kappa = 2$, and in green when $\kappa = 3$. Note that the mapping of the eight possible patterns $\{(000), \dots (111)\}$ onto the vertices of the Boolean cubes is *not* unique; however, different mappings lead to the *same* complexity index κ for all rules, showing that its value is independent of the particular representation chosen.
It might be thought that CA with greater values of κ have also greater computational power, however this is not true. It is true that rules with $\kappa = 1$ can be easily characterized because they describe linearly separable CA, but even rules with $\kappa = 2$ can have extremely complex behaviors as in the case of rule 110, which is known to be equivalent to a Universal Turing Machine. This phenomenon, already hypothesized by Wolfram, is called *threshold of complexity*. It is noteworthy to mention that not all rules with high complexity index have complex behaviors, as it will be show in chapter 6 about *additive rules*.

2.5 A glimpse of physical implementations of cellular paradigms

One of the original purposes of Cellular Neural Networks was performing real parallel computation on-chip. In fact, throughout the last 30 years many CNN implementations have been proposed, and nowadays topographic cell processor arrays are a reality (e.g. [31] [32]). Moreover, the cellular paradigm

has been employed also in cutting-edge general-purpose processors, as already mentioned in the introduction, suggesting that this approach will be widely applied in the near future. The research in this field - especially for vision systems based on the CNN-UM architecture - has been mainly carried out by the Analogic and Neural Computing Laboratory of the Hungarian Academy of Sciences [33], jointly with the Microelectronic Institute of Seville (Spain) [34]. The devices are manufactured and commercialized by Anafocus [35] and Eutecus [36], and they are employed in some of the fastest cameras available on the market.
These devices have the advantages of any special-purpose implementation, like impressive performances and low power consumption, but on the other hand their cost and availability can be an obstacle to their spread use. When a short time-to-prototype is required or in case low cost is an fundamental restriction, it can be better resorting to CNN software simulation [37].
A great compromise in terms of performances and flexibility is given by Graphics Processing Units (GPUs), constituting the core of the last generation of computer graphics cards. Due to their intrinsic parallel hardware and user-oriented software, GPUs combine the positives of hardware implementations and software simulations, and they may become the leading architecture for simulating cellular paradigms.
In the following, we present some of the most-known implementations as well as an outline of the realization of cellular paradigms on GPUs. However, an exhaustive study of the problem goes beyond our purposes, and a deeper analysis can be made thanks to the numerous references provided.

2.5.1 Chips and emulations on reconfigurable devices

Studies about CNN processors trace back to the 90's, when the first prototypes implementing continuous and discrete time CNNs were realized (e.g. [38] [39] [40]). Unfortunately, these devices were not programmable, and the advantages of the parallel computation were overtaken by the delay transfer from the sensors to the processing system.
The invention of smart-pixel chips [41] represented a great improvement in this field, since the capability of processing signals at a focal plane level eliminated the necessity of transmitting large amount of data, one of the bottlenecks of conventional image processing systems. This paradigm has been widely employed in devices based on CNN architecture [5] [42] [43] [44], including modern chips capable of operating with grayscale images at framerates larger than 1000 frames per second [45]. In the last years, the architecture of these devices has tried to resemble the natural vision systems where parallel processing is made at the retina and significant reduction of the in-

formation happens as signals travel from the retina up to the visual cortex; therefore, they combine a first fully parallel analog processing layer, and a second layer composed by digital processors [46]. It is worth to mention that there exist also specific architectures emulating the spatial-temporal dynamic evolutions observed in mammalian retinas [47] [48].
Despite their performances, application specific chips for parallel processing are not extensively used, mainly due to their cost and time-to-market. In order to obtain flexible and inexpensive implementations in a short design time, many authors have emulated cellular architectures on reconfigurable devices, like DSPs and FPGAs . This vein has been particularly successful and it has given rise to dozens of different implementations, whose characteristics are summarized in [49]. As an example, we mention the 'Falcon' [50] architecture, based on a previous implementation called 'CASTLE' [51] [52]. The Falcon processor and its developments have proved to overperform the most advanced general purpose microprocessors on a number of tasks, like the real-time emulation of a digital retina [53]. Remarkably, there are also several examples of CNN systems implemented on FPGA applied to real-life problems [54][55][56]. An alternative implementation on CNNs can be found in [57] as well as its application on the vision system of an autonomous mobile robot [58].

2.5.2 Simulation on Graphics Processing Units

Graphics cards were originally conceived for gaming purposes, but in the last years they have found also scientific applications, especially when it is necessary to implement a truly parallel processing [59] [60]. Having a local network of simple processors, Graphics Processing Units (GPUs) are a natural candidate for simulating the dynamics of CNNs, and a valid alternative to dedicated CNN processors, which tend to be expensive and not always widely available. Simulations on GPUs have the advantage of working on a true parallel hardware, in contrast with other common used software tools [37] [61] that run on traditional sequential computers.
The first studies on CNN simulation through GPUs has recently been presented [62][63], showing promising results. In particular, this solution is as flexible, available, and expensive as a software simulation, but far more efficient. Graphics cards by nVidia seem to be particularly suitable to this application, since they can be programmed through a specific language called CUDA (Compute Unified Device Architecture) [64], whose syntax is very similar to C but especially thought to be employed on multiprocessors units. It should be noticed that the GPU is not used as a general purpose computer, but it only constitutes the processing core of the system. Consequently, the

CPU still accomplishes several tasks, like the transmission of the data to the local memory of the graphics card.
There are several issues that have to be considered when a Cellular Neural Network is implemented on a graphics card, like the divisions of the images into blocks, the handling of variables and data, etc. Often, extracting general rules to set these parameters is not possible, and the best values for them have to be found experimentally.
Nowadays, powerful graphics cards are hosted on most personal computers, and in the short term more software tools will be available to design programs specifically for GPUs. For these reasons, we strongly believe that CNN simulation on GPUs will be an hot topic in the future.

Chapter 3

Alternative proof for the universality of the CNN-UM

Scientists addressed the notion of *computability* even before the invention of digital computers, trying to define *a priori* what classes of problems can be solved by means of algorithms. A thorough analysis of this issue goes beyond the purposes of this dissertation because of its deep mathematical and philosophical implications, however in the following we briefly introduce the concepts of *computability*, *universality*, and *Turing machines* which will be extensively used in the rest of this chapter. In the remaining sections, we present an alternative proof of the universality of the CNN-UM which does not make use of the *Game of Life*. This result has two main consequences: first, it defines a general form for all CNN-UM programs; second, it suggests that the solution space can be explored through a technique called Genetic Programming [65][66]. In this chapter we discuss the first aspect, whereas the second one will be tackled in chapter 4.

3.1 Turing machines, universality, and computability

Turing Machine (TM) [67] is a key concept of theory of computation, and it can be informally described as a system composed by a tape and a head. The tape is *unlimited*, in the sense that it is finite but more paper can be always added, if necessary; it is divided into several cells, each containing a symbol of a finite alphabet, which has to include a special *blank* symbol. The head has an internal state and it can move along the tape; its function is reading and writing symbols in the cells of the tape. Assuming that the TM works in discrete-time, we suppose that at the generic time T the head is on the state

q_i and reads the symbol s_i. Depending on these two values, at the time $T+1$ it changes its state to q_j, writes the symbol s_j (deleting previously s_i), and moves towards the direction d (left or right), like illustrated in the following table

Time	**State**	**Symbol**	**Direction**
T	q_i	s_i	-
T+1	q_j=$q_j(q_i,s_i)$	s_j=$s_j(q_i,s_i)$	d=$d(q_i,s_i)$

This behavior can be conveniently summarized through a so-called *transition function*, which includes as many 5-tuples $< q_i, s_i, q_j, s_j, d >$ as possible combinations of states, symbols, and directions.
After this brief description of its structure, we can give a formal definition [68] of TM as a 7-tuple

$$TM =< Q, \Gamma, b, \Sigma, \delta, q, F > \tag{3.1}$$

where

- Q is a finite set of states;
- Γ is a finite set of symbols;
- $b \in \Gamma$ is the blank symbol;
- $\Sigma \subseteq \Gamma \backslash \{b\}$ is the set of input symbols;
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is a *transition function*, where L (respectively, R) is the left (respectively, right) shift;
- $q \in Q$ is the initial state;
- $F \subseteq Q$ is the set of final states, including an *halt* state.

Note that every part of a TM is finite and discrete, except for the unlimited tape that corresponds to an unbounded amount of storage space. This mechanism is extremely powerful, though simple: it allows the machine to perform computation, and printing the result of its calculation somewhere on the tape. In conclusion, finding an algorithm for doing a problem is equivalent to finding a Turing machine that could solve it [69].
A Turing machine behaves like a computer that executes a given program performing exclusively a specific task. Nevertheless, according to what Turing wrote in [67]

> *[...] it is possible to invent a single machine which can be used to compute any computable sequence. [...]*

which is usually called Universal Turing Machine (UTM). A UTM mimics the action of any other Turing machine, which means that, providing the same input, its output is the same as the original machine. The tape of the UTM contains the transition function of the original Turing machine as well as the input data, both properly encoded.
The importance of the concept of UTM derives from the Church-Turing thesis, according to which

> *Every effectively calculable function is a computable function.*

where in this context "computable" means "produced by a Turing-machine", and "effectively calculable" means "produced by any intuitively 'effective' means whatsoever". In other words, a UTM solves all problems for which an effective method of computation, or algorithm, exists. This property makes the UTM fundamental in the field of computer science, and any paradigm behaving like a UTM is itself called 'universal'.

3.2 Universal CNN models

The Chua-Yang CNN model with a single layer and space-invariant weights is not complex enough to be universal; hence, several authors have proposed other CNN models capable of performing universal computation. Remarkably, a universal CNN can be obtained by making the original model slightly more complex, like adding an extra-layer [70] [71] or a simple nonlinearity [72] [73]. The fact that there is a minimal complexity beyond which a model becomes universal is consistent with the results obtained by Wolfram [28] and Chua [29] about the so-called *threshold of complexity* in one-dimensional Cellular Automata. Other possibilities are using time-varying templates [74], or nonlinear and delayed interactions [75]. The research into this field led to the creation of the Cellular Neural Network-Universal Machine (CNN-UM) [20], already described in Sec. 2.2.
Note that the universality of all CNN models mentioned previously was proved through the *Game of Life*, a two-state two-dimensional cellular automaton equivalent to a UTM [76] [77]. In practice, if a CNN model can run the *Game of Life*, then it is universal. On the one hand, the analogy with the *Game of Life* is widely used, because it is extremely simple to verify; on the other, its relevance is exclusively theoretical, since it does not provide any practical information about the structure of an algorithm performing a

certain task.
In order to find an alternative proof of universality of the CNN-UM, we analyzed other paradigms that are proved to be UTM. The Genetic Programming + Indexed Memory machine (GP+IM machine) is one of those, and because of its importance it is described in detail in the next section.

3.3 GP+IM machine: an example of universal Turing machine

The GP+IM machine was introduced in [78], and it manipulates expressions consisting of terminals (e.g. variables and constants), non-terminals (instructions), and if-then-else statements, which can be conveniently written in form of strings by using the inverse Polish notation (the operator precedes the operands). For example,

$$\mathrm{y} = (*\, 2\, (\mathrm{IF}\ (=\ 3\ \mathrm{x})\ \mathrm{THEN}\ (-\ 1\ \mathrm{x})\ \mathrm{ELSE}(+\ 5\ \mathrm{x})))$$

is a valid expression with six terminals (the input x, the output y, and the constants *2, 3, 1,* and *5*), four functions (**, =, -,* and *+*), and one if-then-else statement. The set of terminals and functions depends on the problem under consideration: for instance, terminals can be real numbers, arrays, images etc.; consequently, functions can be arithmetic operations, like in this example, or any structure manipulating properly the terminals.
This kind of expressions can be easily handled by computer languages like Lisp [79], and they are therefore called "Lisp-like expressions". The name of this paradigm derives from the fact that Lisp-like expressions are also the main elements of Genetic Programming (GP), described in Sec. 4.1, but in this case an Indexed Memory (IM) is also used to store and retrieve values. In order to use the indexed memory we add two new non-terminals in the language: *(Read MP)* returns the value stored in memory position MP, *(Write V MP)* returns the value of memory position MP and change it to V.
The simplest program for the GP+IM machine is called GP+IM function, which is a mapping from inputs to outputs containing only Lisp-like expressions, without iterative or recursive processes. An example of GP+IM function is

$$\mathrm{y} = (\mathrm{IF}\ (=\ 4\ (\mathrm{Read}\ 25))\ \mathrm{THEN}\ (\mathrm{Write}\ 1\ 30)\ \mathrm{ELSE}\ (\mathrm{Write}\ -1\ 30))$$

Lisp-like expressions in general, and GP+IM functions in particular, can be represented by using a class of graphs called Directed Acyclic Graphs

(DAGs) [80], which can be also used to describe CNN-UM functions, as explained in Sec. 3.4.2.
Generic GP+IM programs are obtained by including iterative instructions repeat-until into the Lisp-like expressions. However, as proved in [78], any program for the GP+IM machine can be written in the form

```
Repeat
Evaluate < GP+IM function >
Until
< Some specific state of the memory >
```

Remarkably, this result was not obtained through an analogy with the *Game of Life*, which would have been the typical method, but showing that the GP+IM machine can duplicate the functionality of an arbitrary Turing Machine. Therefore, not only we can affirm that the GP+IM machine is a UTM, but also we know how a generic GP+IM program looks like.
The proof of the universality of the GP+IM machine is obtained by showing that it is always possible to construct a GP+IM machine duplicating the the functionalities of an arbitrary TM. According to the notation introduced in Eq. 3.1, we can suppose that a certain algorithm can be expressed by an arbitrary TM described by the 7-tuple

$$TM =< Q_{TM}, \Gamma_{TM}, b_{TM}, \Sigma_{TM}, \delta_{TM}, q_{TM}, F_{TM} >$$

Now, we want to find a GP+IM machine whose behavior is isomorphic to the one of TM, and expressed by the 7-tuple

$$GP + IM =< Q_G, \Gamma_G, b_G, \Sigma_G, \delta_G, q_G, F_G >$$

where the subscript 'GP+IM' has been shortened to 'G'. As it happens in the definition of Turing Machine, the memory available to TM and GP+IM is unlimited. Here we only sketch the proof, whose complete version can be found in [78].
The proof takes into consideration a GP+IM machine with only three generic operands - X, Y, and Z - which can be either terminals (constants and variables) or sub-functions composed by non-terminals. The set of non-terminals (instructions) contains the following elements:

- (IF X THEN Y ELSE Z):] it returns Y if X is non-zero, otherwise it returns Z;
- (= X Y): it returns 1 if the two arguments are equal, otherwise it returns 0;

- (AND X Y): it returns 0 if either argument is zero, otherwise it returns 1;
- (ADD X Y): it returns the addition of X and Y;
- (SUB X Y): it returns X minus Y;
- (Read X): it returns Memory[X], which is the value of the memory in the position X;
- (Write Y X): it returns Memory[X], and then writes Y in this memory position.

The first step of the proof consists in constructing the elements of the 7-tuple of the GP+IM machine, with a particular emphasis on the transition function. Then, we need to prove the so called ‘equivalence claim’ stating that

> *For any input placed in the GP+IM memory array, the GP+IM machine will accept the input when and only when the target machine accepts the isomorphic input on its tape.*

where a TM can be said to accept its input if it halts in one of the final states In the original paper the authors divide this claim into three short lemmas:

1. All transitions preserve isomorphic equivalence between the two machines;
2. The two machines halt after the same number of transitions;
3. The target machine accepts an input when and only when the GP+IM machine accepts the isomorphically equivalent input;

which are rigorously proved in [78].

3.4 Equivalent ways of describing CNN-UM functions

In general, programs for the CNN-Universal Machine can be represented as a flow diagram containing the following elements:

E1:	global input/output operations,
E2:	CNN operations with specified cloning templates,
E3:	local storage,
E4:	local exchange of information between local memory units,
E5:	local computation combining locally stored values.

We can draw easily an analogy with the GP+IM machine, since also the CNN-UM handles Lisp-like expressions consisting of terminals (global input/output operations, E1), non-terminals (CNN templates, E2), and uses a memory to store and retrieve values (elements E3, E4, and E5). Therefore, by analogy with the GP+IM machine, we define 'CNN-UM function' the simplest kind of CNN-UM program in which the flow diagram contains only if-then-else statements and no iterative process, whereas 'CNN-UM programs' are obtained by including iterative instructions repeat-until. This nomenclature for CNN-UM is not standard, however it helps to notice the similarities between the two kinds of machines taken into consideration.
In the following we analyze four equivalent ways of describing CNN-UM functions: through Universal Machine on Flows algorithms, strings, binary trees, and Directed Acyclic Graphs.

3.4.1 Universal Machine on Flows diagrams

CNN-UM functions and programs are usually described through Universal Machine on Flows (UMF) diagrams [18]. Terminals - which can be logic values and arrays, or analog values and arrays - are represented like in Fig. 3.1(a), whereas the boundary conditions are in Fig. 3.1(b).

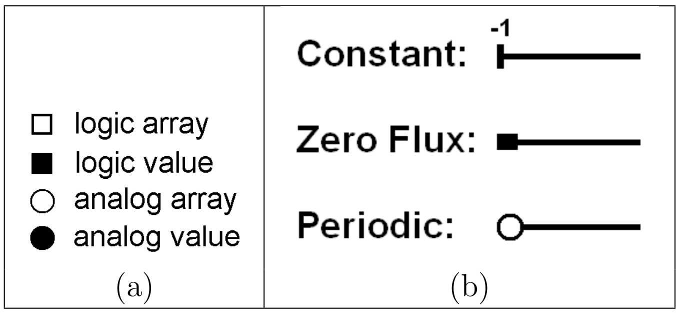


Figure 3.1: Representation for (a) terminals (signals and variables) and (b) boundary conditions in UMF diagrams.

The standard representation for non-terminals (CNN templates) is depicted in Fig. 3.2: Tem$_k$ is the name of the template, whose parameters can be specified explicitly or found in the CNN template library [18]; τ is the time constant; z the bias; U, $X0$ and Y are the input, the initial state, and the output, respectively. Complex functions can be created by combining cascade and parallel structures (Fig. 3.3), and if-then-else statements (Fig. 3.4).

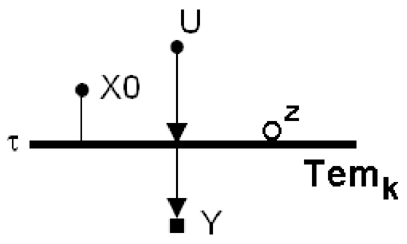
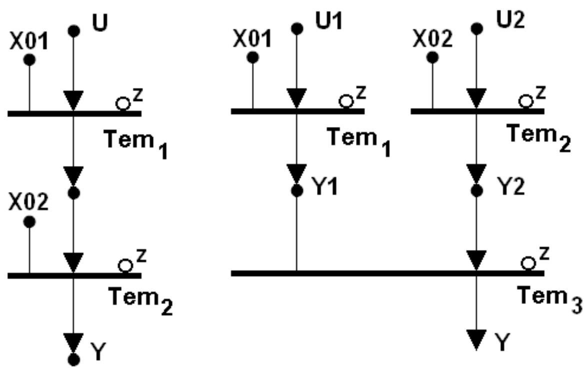


Figure 3.2: Standard representation for CNN templates in UMF diagrams.



(a) (b)

Figure 3.3: Algorithmic structures in UMF diagrams: (a) cascade, (b) parallel.

3.4.2 Strings and binary trees

CNN-UM functions can be also represented by means of strings. In the following, we analyze the case of linear and space-invariant templates, but the conclusions can be easily extended to more general cases.
Following the scheme of Fig. 3.2, we need to specify the input U and the initial state $X0$ for each template, along with other parameters like boundary conditions and integration constant. Therefore, a template Tem$_k$ can be described as

$$\mathrm{Y = Tem_k(U, X0, Params)}$$

This representation can be easily extended to multi-template functions as follows

$$\mathrm{Y = Tem_3(Tem_1(U1, X01, Params1), Tem_2(U2, X02, Params2), Params3)}$$

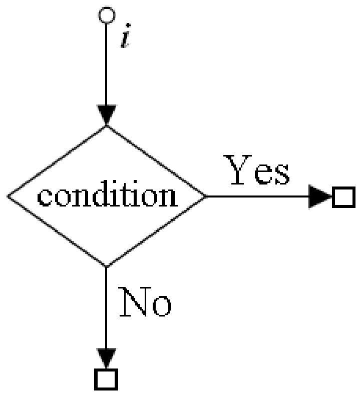


Figure 3.4: If-then-else statement in UMF diagrams.

The input of Tem_3 is the output of Tem_1, whereas its initial state is the output of Tem_2.
Lisp-like functions - and then CNN-UM functions - can be also represented by means of trees, in which leaves correspond to terminals (e.g. input, output, initial state) and branches to non-terminals, such as CNN templates, like in Fig. 3.5.

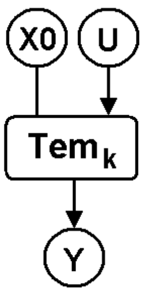


Figure 3.5: Representation for CNN templates in binary trees.

Note that this description focuses exclusively on the structure of the algorithm, without giving any indication about other features like type of terminals or boundary conditions. The input and the initial state of each level of the tree are either outputs of previous levels or terminals (there are no iterative processes in CNN-UM functions); therefore, the first level of the tree is supplied with the inputs, and the output is retrieved from the last level (see Fig. 3.6). Since a CNN template has always two inputs (input and initial state), the resulting trees will be always binary. These two models are equivalent: a binary tree representation can be straightforwardly derived by a string, and vice versa.

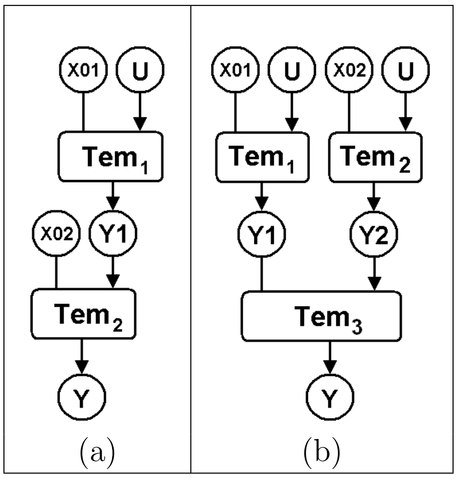


Figure 3.6: Binary trees representing single-template (a) and multi-template (b) CNN-UM functions.

3.4.3 Directed Acyclic Graphs

There is a fourth way of representing CNN-UM functions, which is through Directed Acyclic Graphs (DAGs). DAGs were already mentioned in Sec. 3.3, and in fact they constitute the bridge between the GP+IM machine and the CNN-UM machine, as it will be detailed in Sec. 3.5. In order to explain what a DAG is, we need to define the concept of graph [81].

Definition 3.1 (Graph) *A graph $\mathcal{G}$ is an ordered pair $\mathcal{G}$:=($\mathcal{V}$,$\mathcal{E}$) that is subject to the following conditions:*
1) $\mathcal{V}$ is a set of points, whose elements are called vertices or nodes,
2) $\mathcal{E}$ is a multiset of unordered pairs of distinct vertices (not necessarily distinct), called edges,
The degree of a vertex is the number of other vertices it is connected to by edges.

Definition 3.2 (Directed (Acyclic) Graph) *A graph $\mathcal{G}$ is said to be directed if all edges are directed, where an edge e = (v1, v2) is considered to be directed from v1 to v2; v2 is called the head, and v1 is called the tail of the edge.*
A directed acyclic graph (DAG) is a directed graph with no directed cycles.

A DAG can be considered as a generalization of trees in which certain subtrees can be shared by different parts of the tree. When DAGs are used to describe CNN-UM functions, the vertices with degree 1 are inputs and output, whereas the vertices with degree greater than 1 are CNN templates. In this case, the maximum in-degree of a vertex - that is, the number of edges entering the vertex - is 2 because each CNN template receives one input and one initial state; however, there is no maximum for the out-degree - which is the number of edges exiting the vertex - because the output of a level can be used as input (or initial state) of an arbitrary number of other levels.
A binary tree is a particular case of DAG; moreover, any DAG with maximum in-degree 2, like those representing CNN-UM functions, can be transformed into a binary tree according to the following algorithm:

Algorithm 1 (From DAGs to binary trees) *While there is a vertex $v \in \mathcal{V}$ with out-degree $n > 1$*
1) Make n copies of v, each with the same incoming edges but no outgoing edges;
2) Attach one of the outgoing edges of v to each vertex;
3) Delete v.

3.4.4 Different representations are equivalent

In order to illustrate the equivalence of the four representations illustrated previously, we show how the same CNN-UM function can be described by any of them without loss of generality.
The CNN-UM function considered deletes a selected object [18], and it is fairly simple, containing only five CNN templates: two different shadow templates (*Shadow_HD* and *Shadow_Right*), two kind of edge detectors (*Edge_Top* and *Edge_Left*), plus the *Recall* template; two logic operators (AND and NOT); input (U) and output (Y) images, and no if-then-else- statements. Its UMF diagram is shown in Fig. 3.7, and it is described by the following string

$$\begin{aligned} Y = \quad & AND\,(U, NOT\,(Recall\,(Edge\_Left\,(AND\,(Edge\_Top\,(\\ & Shadow\_HD\,(U))\,, Shadow\_Right\,(U)))\,, U\,))) \end{aligned}$$

The representations of this CNN-UM function through a binary tree and a DAG are illustrated in Figs. 3.8 (a) and 3.8 (b), respectively.

3.5 Analogy between GP+IM machine and CNN-UM

From the results obtained in the previous sections, we can draw a strong analogy between GP+IM machine and CNN-UM machine: in both cases functions can be represented through DAGs, and a number of memories are used to store and retrieve results. Therefore, the results obtained for the GP+IM machine can be automatically extended to the CNN-UM. As a consequence, we can state that the CNN-UM, like the GP+IM machine, is universal; as for our knowledge it is the first time that this result is proved without making use of the *Game of Life*.
Moreover, we can now assert that any CNN-UM program has the following structure

```
Repeat
Evaluate < CNN-UM function >
Until
< Some specific state of the memory >
```

in which the difference between a CNN-UM program and a CNN-UM function is the same as between GP+IM machine and GP+IM function: the former can contain iterative and recursive processes, the latter cannot. In conclusion, a generic CNN-UM is composed by a single repeat-until loop containing a CNN-UM function, which is a combination of CNN templates and nested if-then-else statements.
Although this result gives a general structure for CNN-UM programs, it provides no indication about the details of a CNN-UM program performing a certain task, like what CNN templates need to be used and how the if-then-else statements should be nested. As observed in [78],

> *[...] this notion is similar to saying that Shakespeare is expressible in some language X. This may be good to know, but it does not guarantee that any particular line from language X will be quality literature. Similarly, the language of a typewriter is expressive enough to capture Shakespeare. However, Shakespearean prose will, in practice, never arise from a monkey playing with the keys. [...]*

The techniques that can be used to explore efficiently the space of CNN-UM programs will be discussed in the next chapter.

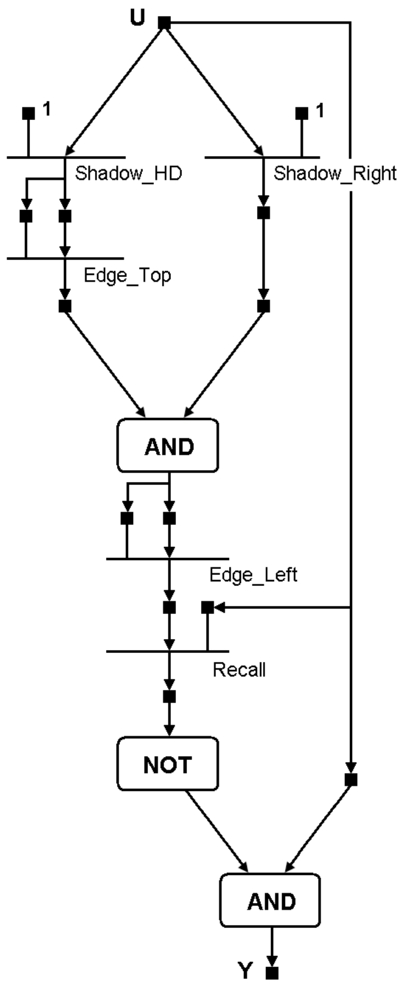


Figure 3.7: UMF diagram of the CNN-UM function to remove a selected object.

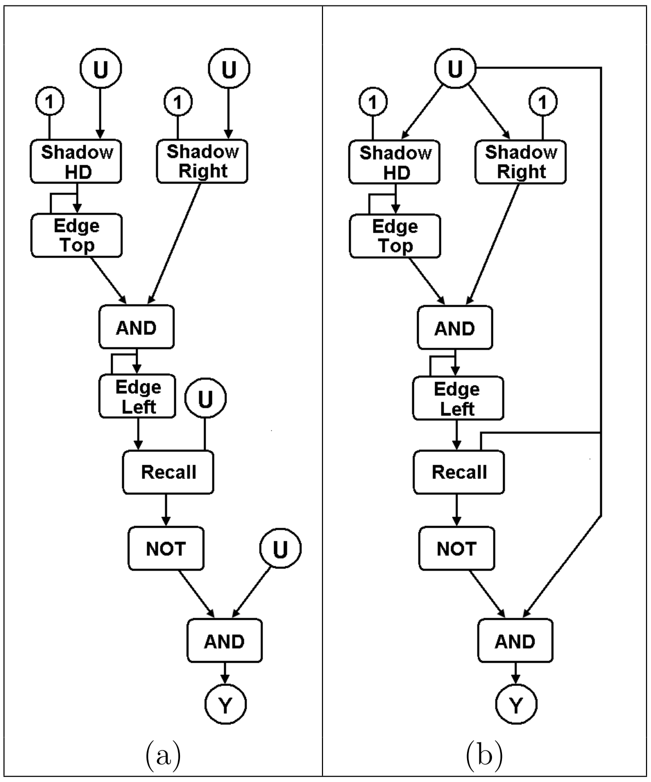


Figure 3.8: (a) Binary tree and (b) DAG of the CNN-UM function to remove a selected object.

Chapter 4

Genetic programming for the CNN-UM

The results presented in the previous chapter define a common structure for all CNN-UM programs, but they do not give any hint about how to choose the CNN templates performing a complex operation and in which order they must be used. In general, there is no systematic way to design CNN-UM programs, and most of the times they are created on purpose for the problem under consideration (e.g., [82][83][84]).
In the past, other authors have investigated the possibility of using machine learning techniques, especially those based on genetic approaches, to explore the search space defined by all CNN-UM programs [85] [86]. Due to the high number of existing CNN templates, such search space is huge and contains all sort of CNN-UM programs, even those with no practical application; without introducing any previous knowledge about the problem, this method is destined to fail. The same system was successively improved [87] allowing the designer to select the CNN templates forming part of the 'candidate' CNN-UM programs evolved by the genetic system. Nevertheless, this change was still not enough to cope with the exploration of the space, and the system either got stuck into local minima, or converged after hundreds of generations even for simple problems, showing little improvement with respect to a mere random search.
Other studies [88] do not focus on CNN-UM programs but on multi-layer CNNs, leaving the choice of the number of network layers to the designer. This system lacks of flexibility, and it suffers from the typical problems of neural network learning. Nevertheless, it achieves some remarkable results in contour extraction and segmentation, successively applied to complex tasks like anti-personnel mine detection [89].
On this ground, we propose a tool combining automatic methods with de-

signer skills to develop effective CNN-UM programs for complex tasks [90] [91] [92]. In particular, we put forward a technique called Genetic Programming (GP), illustrated in Sec. 4.1, to explore the search space composed by CNN-UM programs. It is worth emphasizing that the main difference between our system and the existing ones is that in the former the designer plays a key role, setting many parameters of the system according to the *a priori* knowledge of the problem, whereas in the latter there is practically no place for the algorithm designer.

4.1 Working principles of Genetic Programming

Genetic programming [65][66] is a machine learning technique capable of generating automatically a computer program to perform a given task. It is based on the principles of biological evolution, and its superiority over similar searching algorithms, random search *in primis*, is well-documented in the literature [93][94].
The standard GP flow diagram is shown in Fig. 4.1. The first step to define a GP system is specifying a representation for computer programs (individuals) and an appropriate method to measure their performance (fitness); successively, a set of random programs (population) is created: if it meets a certain requirement (stop condition) then the process stops, otherwise a sequence of operations (generation) is executed. During a generation, the GP system tries to improve the population fitness by combining different programs or creating new ones (genetic operations); these modified programs form a new population (offspring) that replaces the old one (parents). These steps are analyzed in detail in the following, however an exhaustive study of this paradigm goes beyond our purposes.

Fitness function and stop condition

The choice of the fitness function is crucial to the effectiveness of the GP system, and it has to be defined case-by-case depending on the problem analyzed. Usually, several components concur in the definition of the fitness, which are then combined into a unique factor through a weighted sum: for example, the resemblance of the outcome of the algorithm with a reference output; the complexity of the programs (number of instructions or time to execute them) etc.
The population of computer programs is expected to improve - in other words, its fitness is expected to increase - generation after generation until

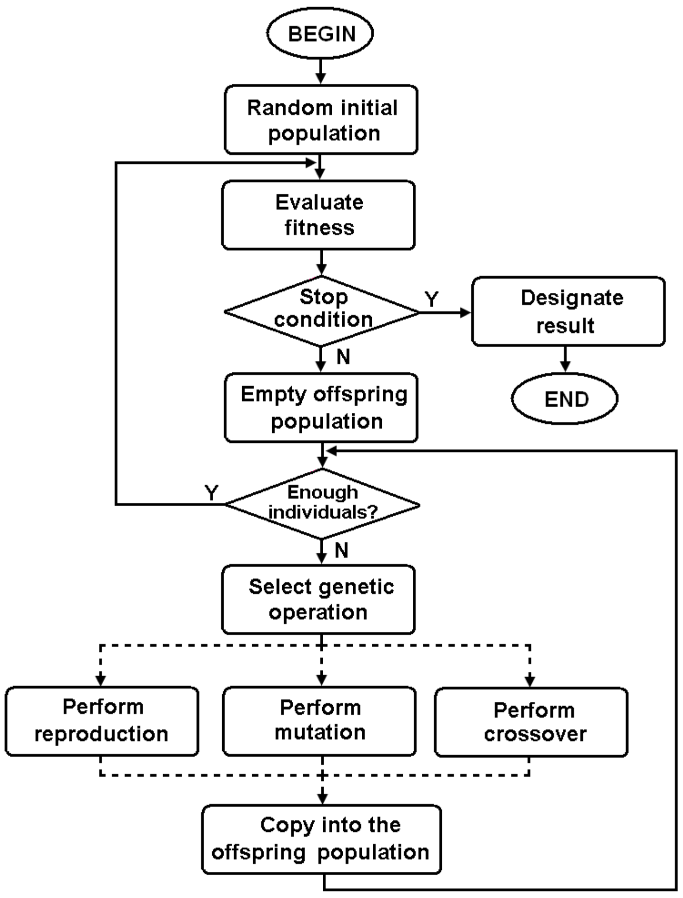


Figure 4.1: GP flow diagram.

the GP halts for having met the stop condition, either because it reached the maximum number of generations allowed or because it found a solution to the problem.

Genetic operators

The number of GP operators is virtually unlimited because new genetic operators can be always be created *ad hoc* for specific cases [95]. In general, they are applied to a subset of individuals probabilistically selected from the population, favoring individuals with higher fitness in order to guarantee that only advantageous features are perpetuated to the following generations. Each operator is employed with a given probability, which can be either fixed *a priori* or varied over the generations. In our system we used only the three

main GP operators: reproduction, crossover, and mutation.
The reproduction is the simplest operator: it copies an individual from the current generation to the following one, assuring a minimal continuity among generations.
The crossover acts on two individuals, selecting a random node in each of them and swapping the subtrees rooted at such positions (see Fig. 4.2); finally, the two new individuals become part of the offspring. Thanks to the crossover, favorable features of different programs can be combined into new individuals.

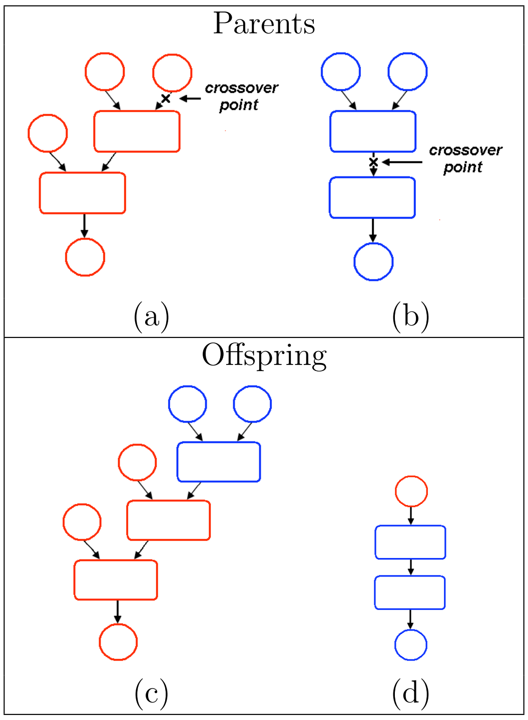


Figure 4.2: Crossover: a random node is selected in each parent (a) and (b), and the resultant 'sub-trees' are swapped to form two new individuals (c) and (d).

The mutation operates on a single individual, choosing randomly a node of the tree: then, either the whole branch from that point downwards is substituted by a new one, like in Fig. 4.3(b), or the instruction corresponding to that node is modified, like in Fig. 4.3(c). The mutation provides diversity to the population, avoiding local minima.

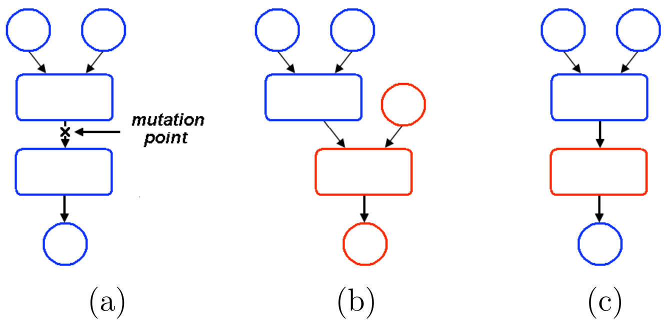


Figure 4.3: Mutation: a point is randomly chosen in an individual (a), then a new tree is created either substituting the whole branch from that point downwards (b), or modifying the instruction corresponding to the node (c).

Population size and number of generations

There are no rules to set the population size and the maximum number of generations of the GP system, and the suitable values for these parameters depend on the nature of the problem. In general, the decision is up to the designer, who determines them according to his or her expertise.
Usually, the population size ranges from few dozens to several hundreds of individuals; there are experimental evidences [96] that a larger population does not imply a faster convergence to the solution, as it might be thought. The number of generations is strictly connected with the population size [97]: in principle, smaller populations take more generations to converge, and vice versa.
In some cases, the GP system does not converge to a solution even evolving a large population for many generations. This may be due to a lack of diversity in the population (number of different individuals) lost - and never recovered - because of an incorrect setting of the operators probabilities; alternatively, the operation set proposed for the experiment may not contain all instructions necessary to represent a solution. In these situations, increasing the number of individuals or generations would not improve the result, but other parameters have to be changed by the designer before repeating the experiment.

Number of runs and premature convergence

It is not infrequent that a large percentage of the population converges to a suboptimal result, causing a drop in the population diversity, and making the creation of new individuals through the crossover very difficult. This phenomenon has a counterpart in nature called *niche preemption principle* [98], which states that

> *[...] a biological niche in nature tends to become dominated by a single species, which may or may not be globally optimal [...]*

In general, the species that dominates a given niche depends on the initial conditions and the history of probabilistic events. The negative effect of premature convergence can be minimized by performing multiple independent runs, using a new random-chosen population at every run. The results obtained on the different runs are then compared, and the individual with best fitness is designated as result of the experiment.
Multiple runs can also measure the amount of computational resource required by the GP solver to yield a success with a certain probability. Be $\mathcal{P}$(M,i) the cumulative probability of success for all generations between 0 and i using a population of size M, and $R(p)$ the number of independent runs required to satisfy the success predicate by generation i with a probability of p = 1-ε; then, the following relation holds [65]

$$\mathcal{P}(M,i) = 1 - 10^{\frac{\log_{10}(\varepsilon)}{R(p)}} \tag{4.1}$$

whose graph is shown in Fig. 4.4. Equation 4.1 allows to calculate the probability of finding the solution using a population size M after i generations.

4.2 A GP system to evolve CNN-UM programs

In our system, the population consists of CNN-UM programs described through any of the representations detailed in Sec. 3.4. Here, we use binary trees in which leaves (terminals) are input and output images, and nodes (nonterminals) are instructions, like CNN templates of if-then-else statements.

The operation set of the GP system contains all instructions and terminals that can be used during the evolution of the population, and it is defined by the designer. It includes the input image(s) and some basic CNN templates (like the 'Threshold' template to binarize greyscale images) as well as any other feature the designer considers appropriate (constants, complex CNN templates, if-then-else statements) taken from [18]. This assures that any *a priori* knowledge about the problem is used profitably, reducing the search space to a manageable size. In principle, there exists always a minimal set of templates suitable for the experiment, or even for all experiments,

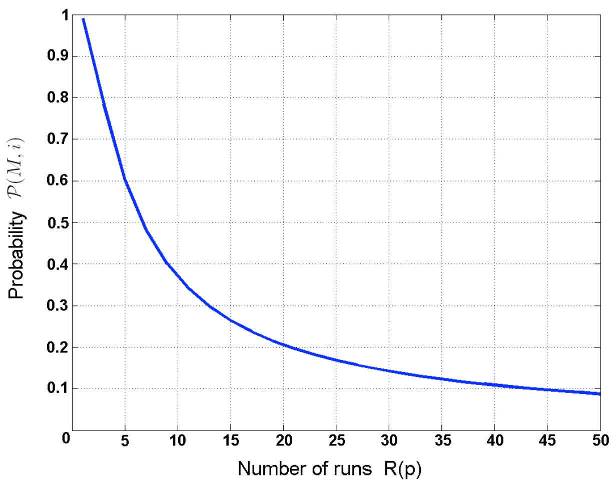


Figure 4.4: Relationship between the average number of runs $R(p)$ and the probability to obtain a solution with a probability p=0.99, given the population size M and the the number of generations i.

because the CNN-UM is universal; this concept is strictly connected with the sufficiency of the system (see Sec. 4.3).

The population of CNN-UM programs is initialized randomly, though considering the information we may have about the algorithm. For example, if we know that a given sequence of instructions certainly appears, we can include it in all individuals; moreover, we can also preserve it from disruptive operations (e.g. mutation) so that it is transmitted unaltered to the following generations. There are several ways to set the initial population, and we compared the performances of three of them: 'full' (all trees have same depth), 'grow' (no fixed depth, each new node is randomly chosen between terminals and non terminals), and 'ramped half-and-half' (an equal number of individuals are initialized for each possible depth, half of the individuals are initialized 'full', and the other half 'grow'). We found experimentally that the 'ramped half-and-half' method produces a more diverse population, giving the best results in terms of average number of generations to converge to the solution.

Our experiments regard image processing problems, then the natural fitness function for our system is a measure of resemblance between the output of a program and the reference image. The choice of the image metric will be discussed in Sec. 4.4. Our fitness function includes also information about the complexity (number of levels and nodes) of the trees: the simpler the tree, the better its fitness.

The only operators available to the GP system are those described in the previous section, which have been applied with the following probabilities: 0.1 reproduction, 0.9 crossover, 0.01 mutation. In theory, an adaptive approach that changes the probabilities at every generation according to a credit-penalty procedure should perform better, but in practice we have not observed any noticeable improvement of the results, despite its computational cost. Note that the system is closed with respect to the operations (see Sec. 4.3), and then crossover and mutation can be performed with any problem.

Our system includes two different mechanisms for the mutation, both applied to a randomly selected individual. The first one acts on the program structure, replacing an arbitrary subtree with a new one; the other one focuses on a single instruction, modifying it partially (e.g., changing the value of the parameter in case of a parametric CNN template) or totally, which means substituting it for another instruction taken from the operation set.

Sometimes, the GP system generates individuals containing redundant subtrees, called introns in analogy with the language of genetics. Whether introns are useful is a long-standing issue: some authors claim that they protect effective subtrees from the destructive action of the operators; others state that introns introduce noise into the system, making the convergence to the solution slower. Probably, the answer to this question depends on the kind of experiments performed. For example, we found experimentally that in our system introns deteriorate the result, hence we implemented a procedure to remove them, as shown in Fig. 4.5.

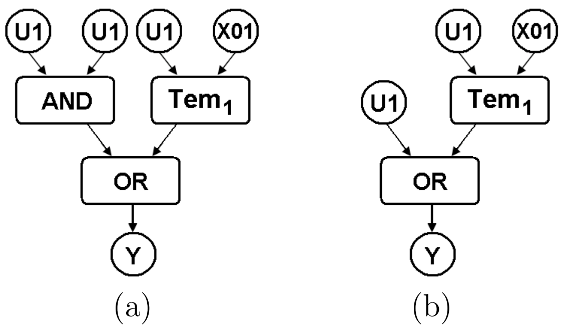


Figure 4.5: The tree in (a) contains a reduntant subtree (intron) that can be simplified (b).

4.3 Closure and sufficiency of the GP system

In any GP system the operation set has to satisfy the conditions of closure and sufficiency [99]: closure means that any individual created during the evolution is valid, in the sense that it can be correctly evaluated; sufficiency implies that the operation set is sufficient to express the solution of the problem analyzed. In the following, we explain how our system meets these requirements.

The first step to prove the closure of the system is verifying that any operation is well-defined for the arguments it may encounter. In our system this is assured thanks to a mechanism very similar to the one used in the CNN-UM. In short, analog and logic values (and arrays) occupy different memory locations, and the instructions can access only to memory slots corresponding to the type of data they handle. For example, CNN templates always use analog and logic arrays to store and retrieve results; if-then-else statements can accept any kind of input, but their output has to be a memory location etc. Note that also the results illustrated in the previous section about the form of the CNN-UM programs are fundamental for proving the closure of the system. We recall that any CNN-UM program can be written as

```
Repeat
Evaluate < CNN-UM function >
Until
< Some specific state of the memory >
```

Therefore, we only need to prove the closure of CNN-UM functions. This result is fundamental, because it means that we do not have to deal with iterations or recursions which would cause problems, as explained in the following example.

Let us consider the two CNN-UM programs in Figs. 4.6(a) and (b), and the offspring obtained by crossing them using the suggested selection points, shown in Figs. 4.6(c) and (d). Note that since the selection point for the tree in Fig. 4.6(b) is inside the backward loop, the structure of the tree in Fig. 4.6(c) is not univocally determined. Should its loop go two templates backward, or rather after the first template of the algorithm? Both cases are compatible with the behavior of the tree in Fig. 4.6(a), and there is no possibility to solve this ambiguity. A similar problem happens for the tree in Fig. 4.6(d), because is this case either the branch goes immediately before the loop instruction (causing an infinite loop), or even before the input, which is absurd. This example shows that the presence of loops does not allow the closure of the system. For this reason, the fact that CNN-UM functions

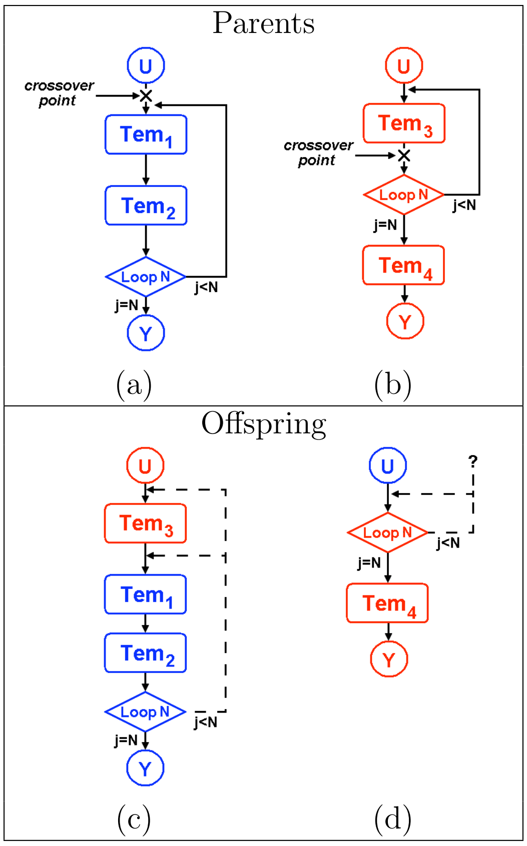


Figure 4.6: Two parents are crossed to obtain an offspring: due to the loops in (a) and (b), the trees in (c) and (d) are not well-defined.

contain only CNN templates and if-then-else statements is crucial, and allows to conclude that the closure property holds.

As for the sufficiency, it is guaranteed by the universality of the CNN-UM (or equivalently, the GP+IM machine). However, the operation set of the GP system is chosen by the designer, who selects the CNN templates that seem promising to solve a given problem. In theory, this may be not a sufficient set, but in practice more CNN templates, terminals, or memory locations, can be added, according to the results of the unsuccessful runs. This process ends in a finite (usually very low) number of steps, after which the designer has selected a suitable operation set for the problem. Obviously, this procedure works when the problem has a solution, or in other words it is computable, according to the definition given in Sec. 3.1.

4.4 Metric functions for images

The choice of an appropriate fitness function is a fundamental aspect of any genetic system because it plays a key role in ranking the individuals. The results of our experiments are images obtained as output of CNN-UM programs, therefore we need to evaluate the similarity between each of these outputs and the reference image. A function calculating the resemblance of two (or more) images is called *metric function*, often shortened to *metric*, and it is defined as follows:

Definition 4.1 (Metric function) *A function d_{AB} of two variables defined on a set $\mathcal{S}$ (metric space) is called a metric function if:*
1. $d_{AB} = d(A, B) \geq 0$, $d_{AB} = 0 \Leftrightarrow A = B$ (Positiveness and Identity);
2. $d_{AB} = d_{BA}$ (Symmetry);
3. $d_{AB} + d_{BC} \geq d_{AC}$ (Triangle Inequality).

In general, there is no metric suitable for all situations: some are very simple to calculate, but they fail to work in non-trivial cases; other succeed in a wide range of different conditions (e.g. they are insensitive to rotations or scale-factors), but they require a huge computational effort. Therefore, the choice of the fitness function is strictly related to the problem considered, and it has to be defined case-by-case.
In the following, we present several metrics that have been used during our experiments. Images are represented by two sets of points denoted by A=(a_1, a_2, ..., a_n) and B=(b_1, b_2, ..., b_n). For the sake of simplicity, we consider linear arrays, in which the index goes from 1 to n; two-dimensional images can be easily represented by this notation. Here, the metrics are defined for binary images (0 for white, 1 for black), because in our experiments we always binarize the input by applying the 'Threshold' CNN template; however, they can easily be extended to greyscale images.

4.4.1 Hamming distance

One of the most used metrics is the Hamming distance ($\mathcal{HM}$), defined as:

$$\mathcal{HM}(A, B) = \sum_{i=1}^{n} |a_i - b_i| \tag{4.2}$$

The Hamming distance measures the number of different points between the two images, and it is equivalent to the pixel-wise exclusive OR (XOR) operation of A and B. On the one hand, it has the indubitable advantage of being simple to implement, but on the other it is strongly sensitive to

noise. Moreover, it does not take into account the shape of the objects, and it cannot be used with scaled or rotated images. To sum up, the Hamming distance is a useful resource when we need a fast evaluation of the fitness function, especially for synthetic images, but it may mislead when applied to real images.
The Hamming distance is symmetrical because it does not make any distinction between reference (A) and test (B) images. In many experiments such distinction is necessary in order to differentiate 'false positives' (1's in the test image corresponding to 0's in the reference image) from 'false negatives' (0's in the test image corresponding to 1's in the reference image): in the first case the test image in noisier than the reference but no information is lost, while the opposite happens in the second case. We can make a further distinction between 'true positives' (1's in the test image corresponding to 1's in the reference image) and 'true negatives' (0's in the test image corresponding to 0's in the reference image), which is useful in some practical situations.

4.4.2 Binary correlation

Another well-known metric is the Binary Correlation ($\mathcal{C}$), defined as:

$$\mathcal{C}(A,B) = \frac{A \cdot B}{n^2 \cdot \sqrt{A^2 \cdot B^2}} \tag{4.3}$$

where n is the array size, and $\cdot$ is the scalar product.
The binary correlation is usually more accurate than the Hamming distance, but also more complex, because of the number of sums and multiplications it involves. Modern machines allow to compute it efficiently, although the performances in terms of speed on parallel hardware are not comparable with those achieved by Hausdorff-like distances, illustrated in next section, which have a direct implementation on CNN-like processors.
The binary correlation, like the Hamming distance, is symmetrical, but in this case it is not easy to weight false positives and negatives. Due to this lack of flexibility, the application of binary correlation in our experiments is limited.

4.4.3 Hausdorff distance and its variations

Some metrics take into consideration the shape of the objects, and they are particularly effective for several classes of problems. As an example, we present the Hausdorff distance ($\mathcal{HS}$) [100] that measures the mismatch between two sets by finding the distance of the point of A that is farthest from any point of B, and vice versa. From a quantitative point of view, the Hausdorff distance can be calculated by using the function $d\mathcal{HS}_{AB}$ - directed Hausdorff distance from A to B - defined as

$$d\mathcal{HS}_{AB} = \max_{a \in A} (\min_{b \in B} ||a - b||) \tag{4.4}$$

In practice, $d\mathcal{HS}_{AB}$ identifies the point $a \in A$ that is farthest from any point $b \in B$, and then it measures the distance from a to its nearest neighbor in B by using some norm $\| \cdot \|$. Analogously, we can define $d\mathcal{HS}_{BA}$ - directed Hausdorff distance from B to A - as

$$d\mathcal{HS}_{BA} = \max_{b \in B} (\min_{a \in A} ||a - b||) \tag{4.5}$$

In general, the directed Hausdorff distance is not symmetric, $d\mathcal{HS}_{AB} \neq d\mathcal{HS}_{BA}$, and then, strictly speaking, it could not be called 'distance'.
The Hausdorff distance $\mathcal{HS}_{AB}$ is the maximum of $d\mathcal{HS}_{AB}$, and $d\mathcal{HS}_{BA}$:

$$\mathcal{HS}_{AB} = \max(d\mathcal{HS}_{AB}, d\mathcal{HS}_{BA}) \tag{4.6}$$

Note that $\mathcal{HS}_{AB} = \mathcal{HS}_{BA}$. Intuitively, if $\mathcal{HS}_{AB} = d$, then each point of A must be within the distance d of some point of B, and vice versa (see Fig. 4.7).
Unfortunately, the Hausdorff distance is extremely sensitive to noise. In order to overcome this drawback, it can be modified by measuring the distance between $A \cup B$ and $A \cap B$, but only in contiguous regions. It is possible to prove that this metric is a nonlinear version of the Hausdorff distance, and hence it is called 'Nonlinear Hausdorff' distance ($nl\mathcal{HS}$) [101], defined as:

$$nl\mathcal{HS}_{AB} = \mathcal{HS}_{AB}(A \cup B, A \cap B)_{\text{except incontiguous regions}} \tag{4.7}$$

A further improvement on the Hausdorff distance is the Integrated Hausdorff distance ($i\mathcal{HS}$), already successfully used in object segmentation and recognition [102]. The Integrated Hausdorff distance is calculated integrating (summing) the Hausdorff distance over all pixels for which the Hamming distance is not zero, that is

$$i\mathcal{HS}(A, B) = \sum_{i,j=1; a_i \neq b_j}^{n} \mathcal{HS}(a_i, b_j) \tag{4.8}$$

In this context, $\mathcal{HS}$ is the local Hausdorff distance, where 'local' means that the Hausdorff distance is calculated for each pixel a_i as if it was the farthest point of the set A. This distance is particularly robust to noise, and it considers the information about the shape of the objects. Furthermore, it can be implemented efficiently on a Cellular Wave Computer [103], but its computation is slow on a traditional computer.

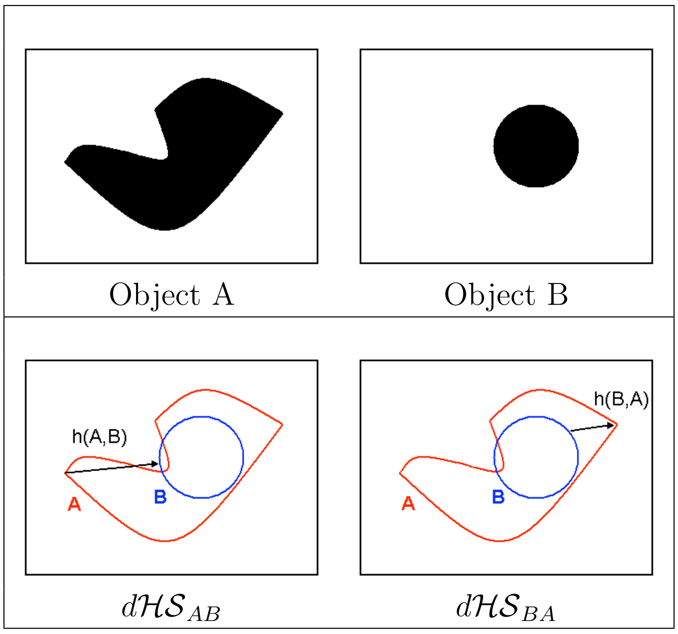


Figure 4.7: Two objects and their direct Hausdorff distances. Note that $d\mathcal{HS}_{AB} \neq d\mathcal{HS}_{BA}$ and $\mathcal{HS}_{AB} = \max(d\mathcal{HS}_{AB}, d\mathcal{HS}_{BA})$.

4.4.4 Fast computation of the Integrated Hausdorff distance

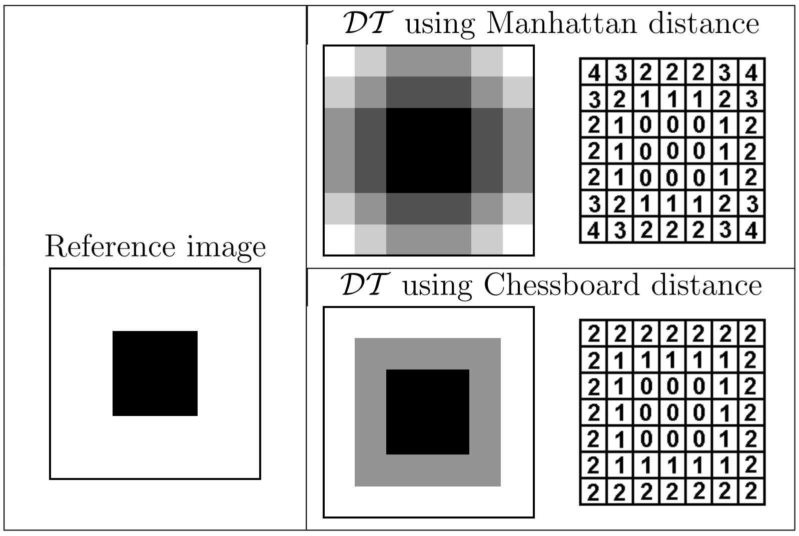
The computational effort to calculate the Integrated Hausdorff distance on a traditional computer can be sensibly reduced by using an image processing operation called *Distance Transform* [104], defined as follows:

Definition 4.2 (Distance transform) *Given a binary image A, the Distance Transform ($\mathcal{DT}$) creates an image A' of the same size as A in which each pixel assumes the value given by its distance (using a certain metric) from the nearest nonzero pixel of A.*

There are two metrics usually employed to compute the Distance Transform: the Manhattan distance, using a 4-connectivity scheme (pixels are connected if their edges touch), and the Chessboard distance, using a 8-connectivity scheme (pixels are connected if their edges or corners touch). For example, the distance between $a_1 = (x_1, y_1)$ and $a_2 = (x_2, y_2)$ is:

$$\begin{aligned} \text{Manhattan distance:} \quad & |x_1 - x_2| + |y_1 - y_2|\,, \\ \text{Chessboard distance:} \quad & max(|x_1 - x_2|\,, |y_1 - y_2|). \end{aligned}$$

An example of computation of the Distance Transform is shown in Fig. 4.8.


$\mathcal{DT}$ using Manhattan distance
Reference image
$\mathcal{DT}$ using Chessboard distance

4	3	2	2	2	3	4
3	2	1	1	1	2	3
2	1	0	0	0	1	2
2	1	0	0	0	1	2
2	1	0	0	0	1	2
3	2	1	1	1	2	3
4	3	2	2	2	3	4

2	2	2	2	2	2	2
2	1	1	1	1	1	2
2	1	0	0	0	1	2
2	1	0	0	0	1	2
2	1	0	0	0	1	2
2	1	1	1	1	1	2
2	2	2	2	2	2	2

Figure 4.8: Distance Transform calculated using two different kinds of distances.

Given a reference image A and a test image B, it is possible to prove that the local Hausdorff distance of a pixel $b_i \in B \Rightarrow b_i = 1$ corresponding to $a_i \notin A \Rightarrow a_i = 0$, is obtained as the value of the Distance Transform of A corresponding to the grid point i. An intuitive explanation of this result can be deduced comparing Fig. 4.8 and Eq. 4.6. Therefore, the Integrated Hausdorff distance between A and B is

$$i\mathcal{HS}(A,B) = \sum_{i=1}^{n} \mathrm{XOR}(A,B). * \mathcal{DT}(A)$$

where $\mathrm{XOR}(A,B)$ is the exclusive OR between the two images, $\mathcal{DT}(A)$ is the Distance Transform of A, and $.*$ is the entry-by-entry product of the two terms.
This method to calculate the Integrated Hausdorff distance is faster than the standard one for two reasons: firstly, the Distance Transform has to be computed only once for each reference image (in most of our experiments we use few reference images), and then the Integrated Hausdorff distance is obtained through a simple multiplication; secondly, our system is implemented in MATLAB which has an optimized function (*bwdist*) to calculate the Distance Transform.

4.4.5 Performance comparison

As we mentioned previously, the choice of the metric is strictly related to the experiment considered; however, some metrics are generally more accurate than others, which means that they perform better in a wide range of situations. Here we compare the metrics described previously an example, taken from [103], which illustrates clearly advantages and disadvantages of each metric.

Differently from what done before, now we deal with continuous functions because they allow to handle integrals and probabilistic distributions. The reference image A for this experiment is shown in Fig. 4.9(a), and it corresponds to the function

$$\xi(x) = k \tag{4.9}$$

whereas all test images are based on the Gaussian distribution $\varphi(x)$ (with mean 0)

$$\varphi(x) = k + \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \tag{4.10}$$

represented in Fig. 4.9(b). For the sake of simplicity, in the following we assume $k = 0$ in our calculations, although images are shown for $k \neq 0$ to avoid clutter.

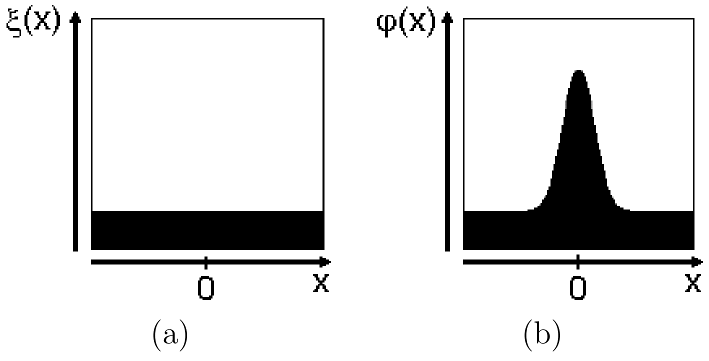


Figure 4.9: (a) Reference function $\xi(x)$, and (b) Gaussian distribution $\varphi(x)$ used for the test.

The Hamming distance between A and the test image B is always equal to the area of the Gaussian distribution

$$\mathcal{HM}(A, B) = \int_{+\infty}^{-\infty} \varphi(x) = 1. \tag{4.11}$$

whereas the Hausdorff and Nonlinear Hausdorff distances have always the same value because objects form only contiguous areas, and it is equal to the maximum assumed by the function $\varphi(x)$

$$\mathcal{HS}(A,B) = nl\mathcal{HS}(A,B) = \max(\varphi(\mathrm{x})) = \varphi(0) = \frac{1}{\sigma\sqrt{2\pi}} \tag{4.12}$$

Now, we examine how the metrics behave in three different test images B, corresponding to a function $f(x)$ obtained by modifying $\varphi(x)$. In the first case, the Gaussian distribution is multiplied by a scalar c, $f(x) = c \cdot \varphi(x)$; in the second case, $f(x) = \varphi(x, \sigma)$, corresponding to variations of the standard deviation σ (B will have always the same area); in the third case, $f(x) = \varphi(c \cdot x)$ (B will have always the same maximum). The results for several values of c and σ, obtained by using Eqs. 4.11 and 4.12, are in Table 4.1; a graphical interpretation is given in Fig. 4.10.

Table 4.1: Functions used in the experiment and its evaluation using different distances.

	$f(x)$	$\mathcal{HM} = \int_{+\infty}^{-\infty} f(x)$	$\mathcal{HS} = \max(f(x))$
1.	$c \cdot \varphi(x)$	$c \cdot \varphi(x)$	$c \cdot \varphi(0)$
2.	$\varphi(x, \sigma)$	1	$\frac{1}{\sigma} \cdot \varphi(0)$
3.	$\varphi(c \cdot x)$	$e^{c^2} \cdot \varphi(x)$	$\frac{1}{\sigma\sqrt{2\pi}}$

In the first case, both $\mathcal{HM}$ and $\mathcal{HS}$ are linear functions of the parameter c; in the second case, $\mathcal{HM}$ is constant and $\mathcal{HS}$ depends on the parameter σ; in the third case, $\mathcal{HM}$ depends on c, whereas $\mathcal{HS}$ does not. However, in the three cases the distance should increase for bigger values of the parameter, because A and B become more different. The results obtained can be explained as follows: the Hamming distance measures only differences in area regardless the shape of the objects (see Eq. 4.11) and then it does not make any distinction between images having equal area, like in the second case; the (nonlinear) Hausdorff distance considers only the farthest points between the two sets, then it fails in classifying images containing peaks with equal value, like in the third example.
Remarkably, the Integrated Hausdorff distance has the desired behavior in the three cases, as shown in Fig. 4.11. This allows us to conclude that, at least in principle, this distance is capable of dealing with misleading situations, and then it is indicated in the most complex experiments.

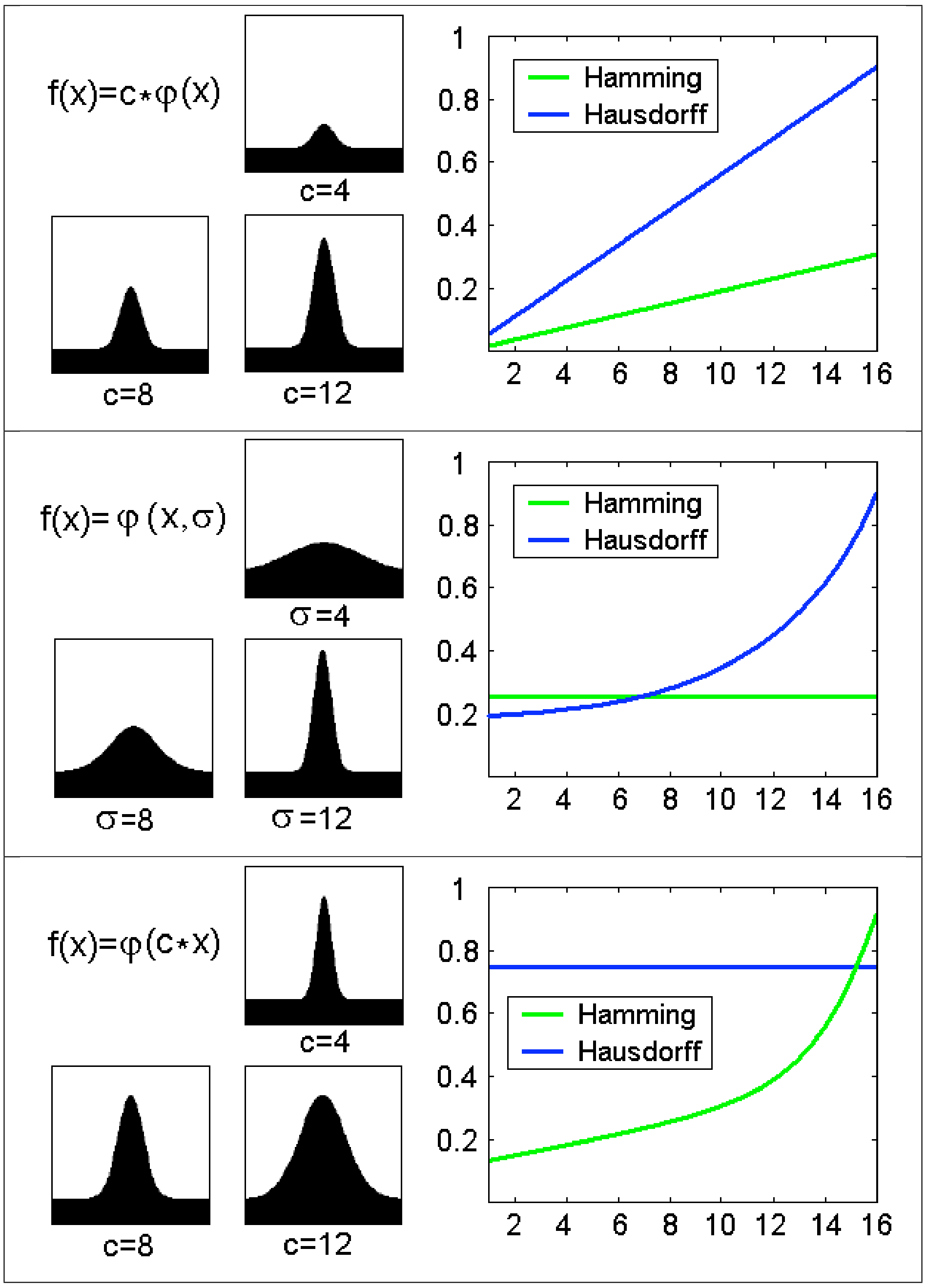


Figure 4.10: Three different experiments: the horizontal axis corresponds to the parameter c or σ, and the vertical axis shows the normalized distances.

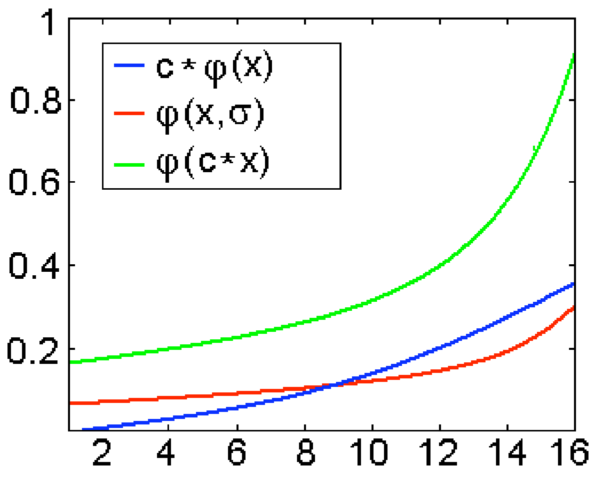


Figure 4.11: Results using the Integrated Hamming distance $i\mathcal{HS}$.

Chapter 5

Experimental results

We have tested our system on several experiments regarding real-life situations taken from the CNN literature. The results, reported in this chapter, are comparable - and in same cases even better - to those presented in the original articles, supporting the effectiveness of our approach. Note that the nature of these problems suggests that a massive parallel computation paradigm, like CNN, can efficiently solve them; however, traditional image processing techniques could have been used too.
Each task is defined through a set of inputs and their correspondent outputs, as usual in a supervised learning framework. Nevertheless, our system does not need a large database of examples as statistical learning methods typically do, but it is able to identify the correct CNN-UM program performing a certain task from a reduced training set. This is because we are not interested in setting the weights of a neural network (we use exclusively standard templates, whose corresponding weights are well-known) but we look how to execute complex tasks by combining elementary ones. Consequently, we perform a sort of morphological learning, in which the system can infer general properties from the characteristics of few representative examples.
The success of our approach relies mainly on the automatic GP-based system, but the human designer still plays a key role by choosing important features like the fitness function, the CNN templates the system can use during the evolution, and the GP parameters (population size, number of generations, runs etc.). Sometimes, the best setting is not achieved at the first attempt, but it has to be found through a trial-and-error process.
The whole system is implemented with MATLAB 7.0, adding two specific toolboxes for Cellular Neural Networks (MATCNN [61]) and Genetic Programming (GPLAB [105]). The time taken by the experiments depends on the setting of the GP parameters and the size of the images, ranging from few minutes to some hours on a 3 GHz Pentium 4 computer.

5.1 CNN template decomposition with restricted weights

In general, hardware CNN implementations accept only a limited set of values for the templates; hence, being able to decompose CNN templates with restricted weights is critical [106]. Here, we take into consideration the *Horizontal Connected Component Detector (HCCD)* template showing how our system is capable of finding a combination of ‘fundamental’ cloning templates (in the sense of CNN templates meeting our restrictions) replicating its functionality.
First of all, we need to identify what CNN model suitable for this case and the form of the matrices A, B, and z. Since the HCCD template detects the number of connected components of each row of the input image, this problem is clearly one-dimensional; therefore, the CNN templates A and B are 1×3 horizontal arrays, and the bias z is a scalar. Moreover, the HCCD template is conveniently performed by a DT-CNN [16] in which at each time step all pixels are moved one position to the right. In this CNN model the input U and the initial state X0 are in practice two different inputs controlled by the matrices A and B, respectively; in the HCCD template the matrix A is set permanently to 0, because there is only one image to process [18]. We can follow the same approach, considering a space-invariance DT-CNN with $A = 0$ and $B \neq 0$, a 1×3 input pattern U, and a scalar output Y. The input U and output Y images follow the standard CNN notation, in which -1 and +1 represent white and black pixels, respectively. To sum up,

$$A = 0, \quad B = (b_{-1} b_0 b_1), \quad z = z_0; \quad U = (u_{-1} u_0 u_1), \quad Y = y_0.$$

There exist several ways of setting the weights of the CNN to realize the *HCCD* template, all leading to the same result, but with different intermediate steps and convergence speeds. They can conveniently described either through the matrices B and z, or, due to the binary nature of input and the output, by means of a truth table in which each combination of U=(u_{-1} $u_0 u_1$) corresponds to a value of Y=y_0. In the following, we consider two different implementations of the *HCCD* template: the first one [107] corresponds to the matrices

$$B_L = (1 \ 1.5 \ -2), \quad z_L = -1.5 \tag{5.1}$$

whereas the values for the second one [16] are

$$B_H = (1 \ 1 \ -1), \quad z_H = 0 \tag{5.2}$$

where the subscripts refer to the initials of the first author. The equivalent truth tables, which will be used as training sets of our system, are in Table 5.1. Note that both mappings lead to the same result; however, it can be easily proved that the the first one converges faster, as confirmed by the example in Fig. 5.2.

Table 5.1: Truth tables for the two implementations of the *HCCD* template.

Input	Training set [107]	Training set [16]
(-1, -1, -1)	-1	-1
(-1, -1, 1)	-1	-1
(-1, 1, -1)	1	1
(-1, 1, 1)	-1	-1
(1, -1, -1)	1	1
(1, -1, 1)	-1	-1
(1, 1, -1)	1	1
(1, 1, 1)	-1	1

Table 5.2: The two implementations of the *HCCD* template converge to the same result, but with different speeds.

Time steps	Training set [107]	Training set [16]
t = 0	■■■□■□	■■■□■□
t = 1	□□■□■■	□■■■□■
t = 2	□□■□□■	□□■□■■
t = 3	□□■■□■	□□■□□■
t = 4	□□□■□■	□□■■□■
t = 5		□□□■□■

As mentioned previously, we want to decompose these templates into combinations of other templates whose elements belong to a given range of values. In this experiment, we set $b_i \in \{0, 1\}$ and $z_0 \in \{\pm 0.5, \pm 1.5, \pm 2.5, \pm 3.5\}$, corresponding to the restrictions of the hardware implementation proposed in [107].
This choice determines also the operation set for our system, which contains all CNN templates combining the 8 allowed values for the triplet $(b_{-1} b_0 b_1)$ with the 8 possible values of z_0; this would give rise to a huge search space formed by 8×8=64 different CNN templates. Nevertheless, it is not difficult

to notice that the behavior of these 64 templates can be summarized by only 10 CNN templates, henceforth 'fundamental' templates, reported in the following by using the notation Tem_k= (($b_{-1}b_0b_1$), z_0):

Tem_1= ((0,0,1), -0.5); Tem_2= ((1,0,0), -0.5); Tem_3= ((0,1,1), -0.5);
Tem_4= ((0,1,1), -1.5); Tem_5= ((1,0,1), -0.5); Tem_6= ((1,0,1), -1.5);
Tem_7= ((1,1,0), -0.5); Tem_8= ((1,1,0), -1.5); Tem_9= ((1,1,1), -0.5);
Tem_{10}= ((1,1,1), -1.5).

In [107] it was proposed a decomposition into 14 'fundamental' templates of the $HCCD$ defined in Eq. 5.1, whereas no solution is known for the the one described by Eq. 5.2.
As for the fitness function to be used by the genetic system, we notice that the best choice is probably a simple Hamming distance, because the problem does not involve any complex images processing requiring a Hausdorff-based metric.
In the first part of the experiment, we evolved 100 individuals over 30 generations, repeating the experiment 200 times (runs). As observed in the previous chapter, these values do not follow any rigorous prescription, but their choice is rather dictated by the designer's expertise and convenience. For instance, in this case the population is large enough to assure a certain diversity among the individuals, but at the same time it is sufficiently small to notice the effects of the evolution on it, since a far larger initial population would probably contain a solution due to the effects of randomness. The best solutions - in terms of number of templates - obtained for both training sets are represented in Figs. 5.1(a) and (b). Remarkably, the tree that our system suggests for the training set in Eq. 5.1 is simpler than the one proposed in the original paper [107], containing only 3 templates instead of 14.
Some interesting conclusions can be drawn by analyzing the number of successful runs (hits) in the two cases: we reported 60 hits with the first training set, and 97 hits with the second one. The cumulative probability of finding a solution, calculated through the Eq. 4.1 (with M=100 and i=30), is P(100,30)=75% in the first case, and P(100,97)=90% in the second case. Obviously, these data suggest that the search case defined by the second training set is easier to explore. This statement is confirmed by the fact that decreasing the population size and the number of generations, the solution for the second training set is still found with high probability. For instance, with only 50 individuals and 10 generations, we obtained 45 hits over 200 runs, resulting in a cumulative probability of P(50,10)=60%.
The second part of the experiment consisted in changing the operation set,

reducing it exclusively to the templates Tem$_1$ and Tem$_2$, because all others are Boolean combinations of these two. The trees describing the solution, shown in Figs. 5.1(c) and (d), are similar to those found with the previous operation set, but now they contain more instructions because of the obvious trade-off between size of the operation set and complexity of the solution.

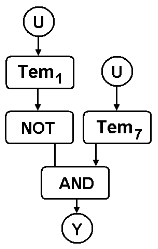
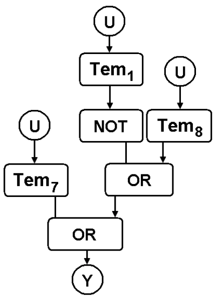
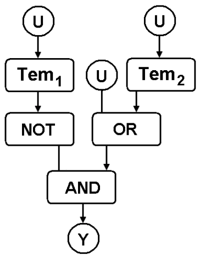
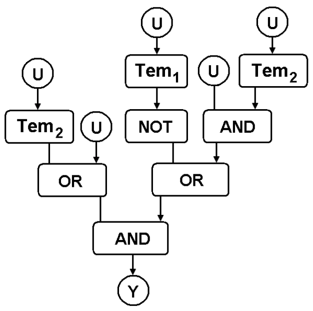
Training set [107]	Training set [16]
 (a)	 (b)
 (c)	 U U U Tem1 U Tem2 Tem2 U NOT AND OR OR AND Y (d)

Figure 5.1: Best solutions for both training sets; the two rows correspond to different operation sets.

5.2 Roughness measurement

A problem arisen in a real-life application was distinguishing engine debris from gas bubbles in helicopter gasoline: the former may damage irremediably the motor, whereas the latter are innocuous. A successful approach to tackle this task was suggested in [84], which consists in measuring the object roughness by finding its edge concavities: approximately rounded objects are considered to be bubbles, all others, showing all kinds of irregularities, are debris. A CNN-UM algorithm solving the problem is depicted in Fig. 5.2: it is very effective despite its simplicity, as shows the example in Fig. 5.3.

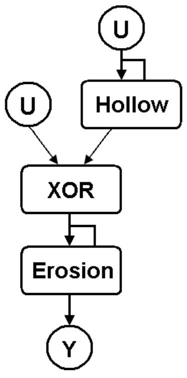


Figure 5.2: CNN-UM algorithm measuring the roughness of an object proposed in [84].

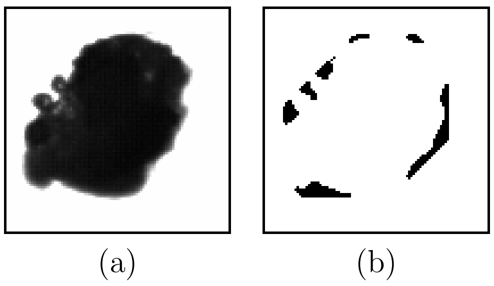


Figure 5.3: Input image (a) showing a debris and output (b) obtained thanks to the algorithm proposed in [84]. These same images have been used as training set of our genetic system.

Now, we want to discover whether our system is capable of finding a solution - either the same as [84] or a new one - to this problem.
In this experiment we use a training set composed by one pair of input-output images shown in Fig. 5.3. Although it may seem too small, this reduced set of examples is sufficient to train properly the system, due to the 'morphological learning' performed by our system, completely different from a statistical learning used for a traditional neural network.
As for the fitness function, we use the Integrated Hausdorff distance (Sec. 4.4), which was created on purpose for this experiment [84]. The choice is dictated by the nature of the problem, in the performances of the system are

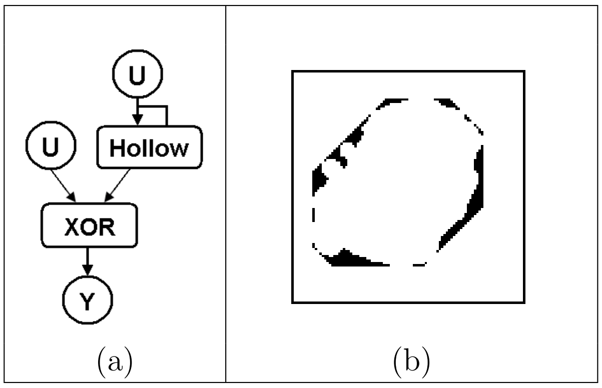


Figure 5.4: Roughness measurement: the first tree (a) proposed by the system, corresponding to the output in (b) which differs slightly from the the one in Fig. 5.3(b).

evaluated thanks to considerations on the shape of the objects.
The operation set should contain the main CNN templates performing morphological operations, like the *Edge* template, detecting the edge of the objects; the *Hollow* template, which fills the hollow parts of the objects; the *Figure extraction* template, extracting all connected figures in the input image corresponding to black pixels in the initial state image; and the *Find area* template, which finds solid black framed areas in the image. We include also the *Threshold* template - which transforms the input greyscale image into a binary one - as well as basic logic operations like AND, OR, NOT, and XOR. The final operation set is therefore: $\{Edge, Hollow, Figureextraction, Findarea, Threshold, AND, OR, NOT, XOR\}$.
The complexity of the problem, at least in principle, is similar to the one of the previous section, hence also the parameters for the GP system should be approximately the same. Therefore, we evolved 100 individuals during 25 generations, but it turned out that in none of the runs the system was able to find a solution matching exactly the desired output; the CNN-UM program with best fitness is depicted in Fig. 5.4(a), whereas its output Y is in Fig. 5.4(b).
Although the output obtained is substantially correct, it is still necessary to remove small imperfections to get exactly the desired result. Therefore, we include the *Erosion* template - which makes this kind of small refinements - in the operation set, removing at the same time the *Figure extraction* and the *Find area* templates, which have not been used in the solution; hence, the final operation set is: {Threshold, Hollow, Erosion, XOR, AND, OR, NOT}. Note that the search space is now smaller than before because there

are fewer available templates (7 vs. 9); consequently, also the population size and the number of generations can be reduced. Evolving only 50 individuals over 15 generations, we obtain the solution at practically every repetition of the experiment. Remarkably, our system finds not only the CNN-UM program proposed in [84], reproposed in Fig. 5.5(a), but also an alternative one, shown in Fig. 5.5(b). This confirms that this approach helps to find innovative solutions, not always evident to the human designer.

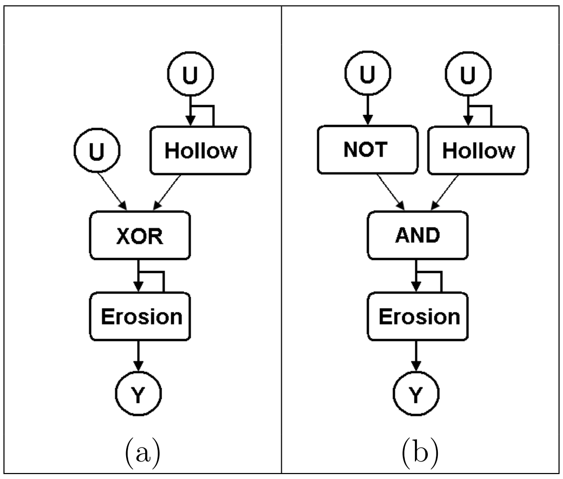


Figure 5.5: The two solutions found by our system for the roughness measurement.

5.3 Texture segmentation

The CNN templates listed in the standard library [18] can solve *many* but not *all* problems. Sometimes, it is necessary to create *ad hoc* a CNN suitable for a certain task, as for example in [108], presenting a CNN-UM program to perform texture segmentation that includes a CNN template expressly designed for this experiment. The main drawback of this CNN template is that it contains double-digit decimal values that make it very sensitive to noise when implemented on hardware devices. Moreover, it has been designed thanks to a genetic-based method based on statistical learning: it means that many examples have been shown to the system, which set the weights of the Cellular Neural Network accordingly. This template might not work with other examples too, but this depends on how representative the training set is.

In this section we show how our system is capable of decomposing this

specific-purpose CNN template into a sequence of standard CNN templates. Such simplification gives two main advantages: first, standard templates are usually robust, and their optimal implementation on CNN hardware is well-known; second, we do not need a complex training set, but we rely entirely on one example, showing that the results found are valid for all others.
The pair of input-output images defining the task are displayed in Figs. 5.6(a) and (b). They represent two different textures (one horizontal, and the other vertical), whose interlacement forms the word 'CNN'.

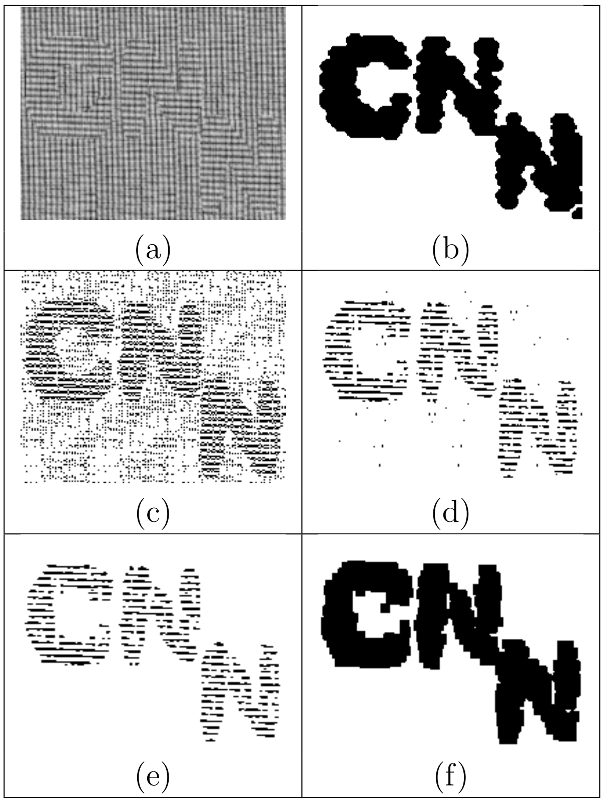


Figure 5.6: Texture segmentation: (a) input image, (b) result found in [108], (c) *Vertical Line Remover*, (d) Erosion, (e) *Horizontal Erosion*, (f) output of our algorithm.

According to what said in Sec. 4.4, the fitness function for this experiment should be based on the Hausdorff distance, because we need to take into consideration the shape of the objects. Unfortunately, this is not possible because the input and output images used are quite big (192×154 pixels), and the evaluation of the individuals with a Hausdorff-based fitness would be excessively slow. As a consequence, we resort to a Hamming-based fitness,

modifying the standard Hamming distance to make a distinction between false negatives and false positives. The presence of 'holes' in the figure (false negatives) would diminish the intelligibility of the word 'CNN', whereas a black noise (false positives) would not have a negative effect (within certain limits) because the contours of the letters hidden in the texture are not well-defined. Consequently, we assign a greater weight to false negatives than to false positives.
As usual, the operation set contains the CNN templates that seem be appropriate for this experiment. In this case we include standard logic operations - like AND, OR, NOT and XOR -, templates to remove noise from images - like *Erosion*, *Dilation*, *Isolated point remover*, *Vertical erosion*, and *Horizontal erosion* - and templates to eliminate horizontal and vertical lines (*Horizontal line remover* and *Vertical line remover* templates), because we need to discern horizontal and vertical lines in the textures. The details of these templates can be found in [18], except for the *Horizontal erosion* template whose parameters are:

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix} ;\, B = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} ;\, z = -4.$$

Note that the *Vertical erosion* template can be obtained by rotating the B matrix. We also combined the *Dilation* and the *Erosion* templates into a subroutine called *Closing*. Tu sum up, the operation set employed is: {*AND, OR, NOT, XOR, Closing, Isolated point remover, Vertical erosion, Horizontal erosion, Horizontal line remover, Vertical line remover* }.
The task is definitely more complicated than the ones presented in the previous sections, then we increased the population size to 150 individuals evolving them over 30 generations. The algorithm obtained is depicted in Fig. 5.7, where letters within brackets refer to the correspondent images shown in Figs. 5.6(c) to (f). Comparing the result obtained (Fig. 5.6(f)) with the one proposed in the original paper (Fig. 5.6(b)), we can conclude that the two solutions are very similar, and both allow to identify the word hidden in the texture. Our algorithm, though, has the advantage of using only standard templates, and then being more robust and implementable easily on real devices.

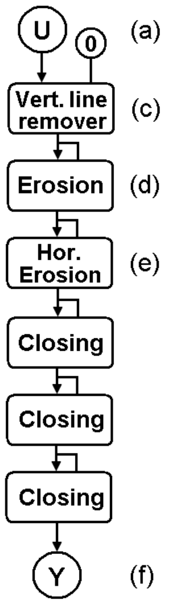


Figure 5.7: Algorithm for the texture segmentation. The letters within brackets refer to Fig. 5.6.

5.4 Route number localization on public transport vehicles

Cellular Neural Networks have been successfully employed in a number of real-life applications like the Bionic Eyeglass [109], an ambitious project to create a device to aid visually impaired people in their daily life. The Bionic Eyeglass consists in a portable camera based on the CNN technology, which can efficiently perform image processing algorithms. The challenge is defining CNN-UM programs that can be used in real scenarios faced by visually impaired people, like crosswalks identification, banknote recognition etc. Obviously, the CNN-UM programs have to be robust, since they have to work within a wide range of conditions, and simple enough to be executed in real-time.

Here we tackle the problem of localizing the route number sign of a public transport vehicle on a low-resolution monochrome image. In particular, we consider black signs on a white background, corresponding to a realistic case in many cities (e.g. Budapest). The images to train and test our system were taken with a camera of a cell phone under different conditions of brightness; the goal is identifying reliably the sign in the figure, like the number '19' in the image of Fig. 5.8(a). This is the only image used to train our GP system, whereas other images showing different vehicles will be employed to validate the result obtained. The fact of using such a reduced training set shows how easily a morphological learning technique can generalize.

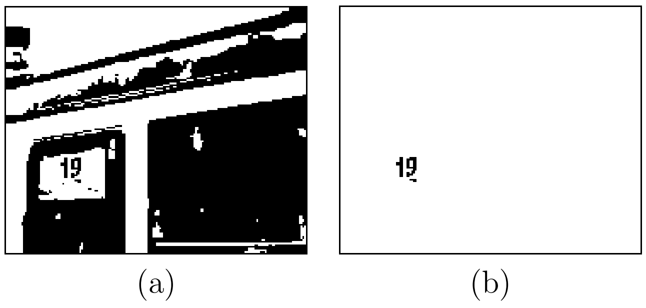


Figure 5.8: Input (a) image for the rout number localization, and output (b) obtained by our system.

The considerations about the fitness function are similar to those made in Sec. 5.3. We would like to use a Hausdorff-based metric, but the size of the input and output images (122×84 pixels) would make this process very time-consuming; therefore, we resort to a modified Hamming-distance. Also in this example we prefer to have a slightly noisy output (false positives) rather than removing parts of the numbers in the image (false negatives), which would make the number recognition task extremely difficult. In other words, false positives do not influence the final result as much as false negatives, and then we changed the fitness function consequently.
The operation set contains the CNN templates that seem to be useful for this experiment. We included basic logic operations (AND, NOT, OR, XOR); a template to make a logic difference between two binary images (DIF template); the *Closing* routine, which remove noise by dilating and shrinking the image; the *Edge* template, detecting the edges of the objects; the *Find area* template, which finds solid black framed areas in the image; the *Hole* template, filling the holes in the image; and finally the *Recall* template, recalling the objects marked by black pixels. To sum up, the operation set for this experiment contains 10 CNN templates, namely {*AND, NOT, OR, XOR, DIF, Closing, Edge, Find area, Hole, Recall*} , all coming from the standard CNN template library [18]. Note that the greyscale input images need to be thresholded in order to transform them into binary.
We evolve 100 individuals over 30 generations, limiting the maximum number of levels of the tree to 20 (because of the complexity of the problem, the system may tend to use excessively big trees, whose fitness evaluation is

time-consuming).
The CNN-UM program found automatically by the GP system is depicted in Fig. 5.9(a), and the output obtained is shown in Fig. 5.8(b). It is noteworthy that the resulting tree contains only 7 levels between the input image U and the output image Y.

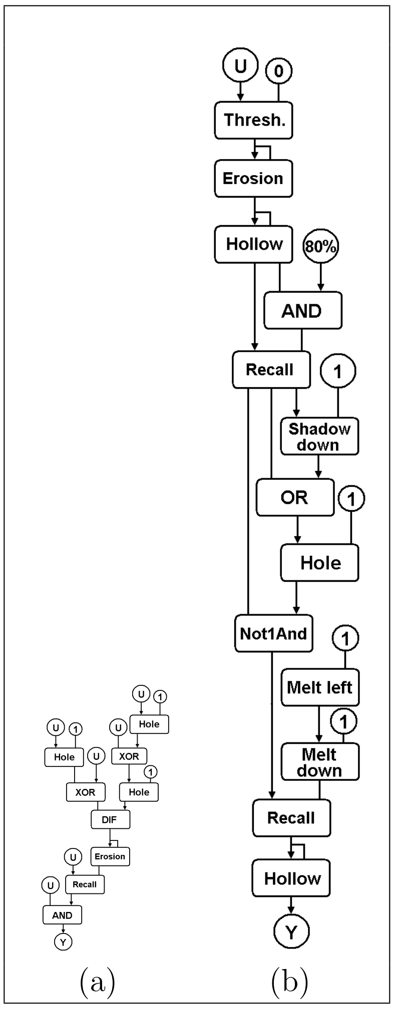


Figure 5.9: Route number localization: solution found by the automatic system (a) and the one presented in the original paper (b).

The generalization capability of our CNN-UM program can be testing it on two validation examples, representing different scenarios. The input images are in Figs. 5.10(a) and (c), and the output obtained are in Figs. 5.10(b) and (d), respectively. In both cases our algorithm detects the route number correctly, and the small residual noise, which can be noticed especially in

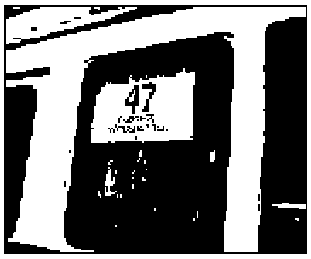
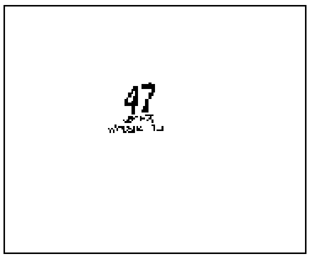
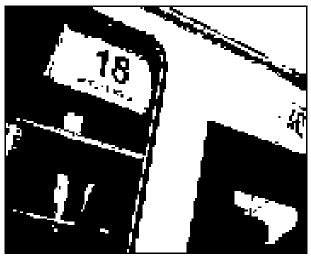
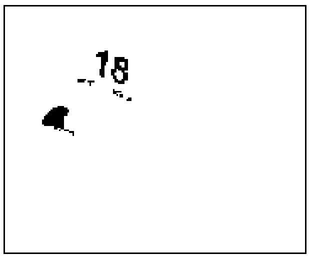
Input	Output
 (a)	 (b)
 (c)	 (d)

Figure 5.10: Validation of the algorithm for the route number localization on two different input images (a) and (c); the results are in (b) and (d), respectively.

Fig. 5.10(d), can be easily removed by a further post-processing.
With respect to our algorithm, the one presented in the original paper [109] and shown in Fig. 5.9(b), contains more levels (12), non-standard templates (e.g. *Melt left* and *Melt down*), and complex terminals, like '80%' which is a vertical line with a certain length. We can then conclude that the solution found through automatic methods is better than the original one, both in terms of complexity (number of templates) and robustness (standard templates and terminals), and its execution on a real device - still to be done - should be faster and more reliable.

Chapter 6

Nonlinear dynamics perspective of Cellular Automata

Cellular automata (CA) have been known for decades, since von Neumann conjectured that interdependent elementary structures can generate complex dynamics [27]. In fact, despite their apparent simplicity, CA have not been completely characterized yet, but they have already found application in a variety of fields (e.g., physical [110] and biological [111] systems modeling, social sciences [112], and game theory [113]).
Researches have been particularly active after the publication of Wolfram's works [114] [28], which formalized many properties of CA, proposed a classification of local rules based on empirical criteria, and proved that even a one-dimensional two-state Cellular Automaton is indeed universal.
In the very last years, Chua has addressed the problem by using the theory of nonlinear dynamics [29]: thanks to this insight into CA, it has been possible to define new concepts, like the complexity index of a CA rule, and give a rigorous explanation of well-known properties, such as the global equivalence of local rules. Moreover, Chua advanced a new classification of CA rules, based on the properties of the cycles obtained by evolving finite length bit strings under local rules.
In this dissertation we summarize this new perspective of Cellular Automata, introducing fundamental concepts in the first part, which will be used in the next chapter to prove some important practical consequences.

6.1 Cellular Automata are a special case of Cellular Neural Networks

As proved in [115], it is possible to design a CNN cell, defined by a scalar nonlinear differential equation, whose output tends to an attractor that codifies *any* binary CA local rule, independently of the sphere of influence r and the spatial dimension. This result can be particularized for $r = 1$ (equivalent to the nearest neighbors case) and one-dimensional structures, as summarized by the following theorem

Theorem 6.1 *A one-dimensional nearest neighbors binary cellular automaton is a special case of a linear CNN with the same neighborhood size.*

Therefore, each CA local rule can be mapped into a nonlinear dynamical system whose attractors encode the associated truth table (see Table 2.1). The scalar differential equation generating the 256 rules of a one-dimensional binary Cellular Automaton is [116]

$$x_i^{n+1} = \mathrm{sgn}\{z_2 + c_2|(z_1 + c_1|z_0 + b_1x_i^n + b_2x_{i+1}^n + b_3x_{i-1}^n i + 1|)|\} \qquad (6.1)$$

where the choice of the eight parameters $\{z_2, c_2, z_1, c_1, z_0, b_1, b_2, b_3\}$ determines the CA local rule $\mathcal{N}$ performed (see Table 4 in [116]). Note that each realization of $\{z_2, c_2, z_1, c_1, z_0, b_1, b_2, b_3\} \in \mathbb{N}^8$ defines one rule, but there are infinite combinations of the parameters $\{z_2, c_2, z_1, c_1, z_0, b_1, b_2, b_3\}$, corresponding to regions of the space $\mathbb{N}^8$, realizing a local rule. As found experimentally, the choice of the parameters is quite robust for any of the 256 CA local rules, and hence the regions of the space are relatively large.
To sum up, Eq. 6.1 describes a CNN cell (neuron) performing *all* CA rules, including the *universal* rule 110, and then it is called *universal neuron.* From the information theory point of view, the universal neuron is also a minimal representation, since the number of its parameters (8) is equal to the number of bits describing a CA rule.

6.2 Visual representations for CA local rules

6.2.1 Time-τ characteristic function

We recall from Sec. 2.4 that in a one-dimensional CA with $L = I + 1$ cells there are 2^L different Boolean strings, constituting the so-called 'state space' Σ. A generic CA rule $\mathcal{N}$ induces a global map

$$T_{\mathcal{N}} : \Sigma \to \Sigma \qquad (6.2)$$

where each string $\mathbf{x} \in \Sigma$ is mapped exactly into another through the transformation $T_{\mathcal{N}}(\mathbf{x}) \in \Sigma$, and this mechanism allows the CA to evolve throughout time. For instance, given the bit string $\mathbf{x^n} = (x_0^n x_1^n \dots x_I^n)$ at time n, the bit string $\mathbf{x^{n+1}} = (x_0^{n+1} x_1^{n+1} \dots x_I^{n+1})$ at time $n+1$ is obtained as

$$\mathbf{x^{n+1}} = T_{\mathcal{N}}(\mathbf{x^n}) \tag{6.3}$$

To each string $\mathbf{x} = (x_0 x_1 \dots x_I)$ we can also univocally associate a real number $\phi(\mathbf{x}) \in [0, 1)$ by means of the following transformation

$$\phi(\mathbf{x}) = \sum_{i=0}^{I+I} x_i \cdot 2^{-(i+1)} \tag{6.4}$$

For example, if $\mathbf{x} = (0 \dots 0)$ then $\phi(\mathbf{x}) = 0$, whereas if $\mathbf{x} = (1 \dots 1)$ then $\phi(\mathbf{x}) = 0.99 \dots 9$, independently of the length L. It is possible to prove [117] that if $I \to \infty$ then $\phi(\mathbf{x})$ tends to assume *all* rational values belonging to the interval $[0, 1)$, which, as well-known, are a dense subset of the real numbers; hence, we use the notation ϕ to indicate the whole horizontal axis $[0, 1)$. Equation 6.4 induces a global map similar to Eq. **??** called *time-1 characteristic function* defined as

$$\chi_{\mathcal{N}} : \Re[0, 1) \to \Re[0, 1) \tag{6.5}$$

and

$$\chi_{\mathcal{N}}(\phi) = \sum_{i=0}^{I} T_{\mathcal{N}}(x_i) \cdot 2^{-(i+1)} \tag{6.6}$$

where $T_{\mathcal{N}}(x_i)$ stands for the i-th element of the array $T_{\mathcal{N}}(\mathbf{x})$. Note that the characteristic function is an alternative way to define a rule, and its graph can be easily drawn as follows:

1. Divide the unit interval $\phi = [0, 1)$ into a finite number of uniformly-spaced points of width $\Delta\phi$;

2. For each grid point $\phi(\mathbf{x}) \in [0, 1)$ identify the corresponding binary string $\mathbf{x} \in \Sigma$;

3. Determine the image $T_{\mathcal{N}}(\mathbf{x})$ via the truth table of $\mathcal{N}$;

4. Calculate the decimal equivalent $\chi_{\mathcal{N}}$=$\chi_{\mathcal{N}}(\phi(\mathbf{x}))$ of $T_{\mathcal{N}}(\mathbf{x})$ through Eq. 6.4;

5. Plot the point ($\phi(\mathbf{x})$,$\chi_{\mathcal{N}}$);

6. Repeat these steps over all $\frac{1}{\Delta\phi} + 1$ points.

The reason why this kind of function is called *time-1 characteristic function* $\chi^1_{\mathcal{N}}(\phi)$ is that it considers only one iteration of the CA, i.e. $\mathbf{x^n}$ and $\mathbf{x^{n+1}}$ (or equivalently, $\phi^n = \phi(\mathbf{x^n})$ and ϕ^{n+1}=$\phi^{n+1}(\mathbf{x^{n+1}})$). However, important properties of CA can be discovered by analyzing the *time-τ characteristic function* $\chi^\tau_{\mathcal{N}}(\phi)$, which takes into consideration a bit string $\mathbf{x^n}$ and the output $\mathbf{x^{n+\tau}}$ obtained by iterating it τ times. The corresponding decimal values of these bit strings can be used to draw the graph of $\chi^\tau_{\mathcal{N}}(\phi)$ according to the algorithm illustrated before. An example of time-1 and time-2 characteristic functions for rule 150 are shown in Fig. 6.1(a) and (b), respectively. Note that in general we omit the superscript for the time-1 characteristic function.

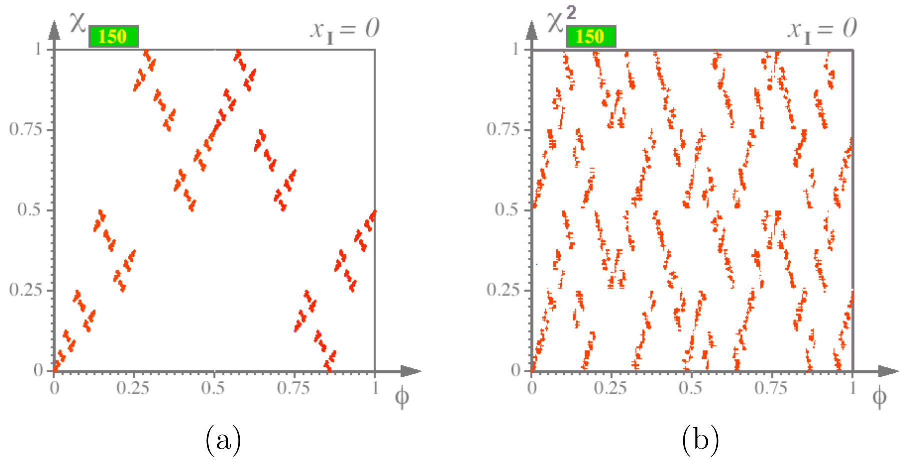


Figure 6.1: (a) Time-1 (a) and (b) time-2 characteristic functions for rule 150. To avoid clutter, only the case for $x_I = 0$ is shown.

6.2.2 Time-τ return plot and Lameray diagram

Another useful way to visualize CA local rules is the time-1 return plot $\phi^1_{\mathcal{N}}$, which focuses on the evolution of a specific bit string throughout time. In practice, the time-1 return plot can be defined as the set of bit strings obtained evolving a specific bit string $\mathbf{x^0} = (x^0_0 x^0_1 \ldots x^0_I)$ under a local rule $\mathcal{N}$

$$\phi_{1,\mathcal{N}}(\mathbf{x^0}) = \{T_{\mathcal{N}}(\mathbf{x^0}) \;\; T^2_{\mathcal{N}} \ldots\} \tag{6.7}$$

where

$$T^2_{\mathcal{N}} = T_{\mathcal{N}}(T_{\mathcal{N}}(\mathbf{x^0})).$$

The time-1 return plot can be conveniently represented in a Cartesian coordinate system, where on the abscissa axis there is the parameter $\phi_{n-1} = \phi(\mathbf{x^{n-1}})$ and on the ordinate axis the value $\phi_n = \phi(\mathbf{x^n})$ (equivalently we can use ϕ_n and ϕ_{n+1} on the abscissa and the ordinate, respectively). An example of time-1 return plot for rule 62 is in Fig. 6.2(a); note that in this example we have chosen a value of $\mathbf{x^0}$ such that $\mathbf{x^0} = T^3_{62}(\mathbf{x^0})$, and then the time-1 return plot contains only three points.
The *qualitative* dynamics of a local rule $\mathcal{N}$ can be alternatively represented through Lameray diagrams [118], also called cobweb diagrams [119]. In a certain sense, their essence is similar to time-1 return plots because they follow the evolution of a single bit string $\mathbf{x^0} = (x^0_0 x^0_1 \ldots x^0_I)$ throughout time, and it is plotted in a (ϕ_{n-1},ϕ_n) Cartesian coordinate system. However, there are two main differences with respect to time-1 return plots: first of all, in this case we draw not only the points, but also the lines connecting them; second, in Lameray diagrams we discard the *transient*, hence the circuit depicted will always be closed.

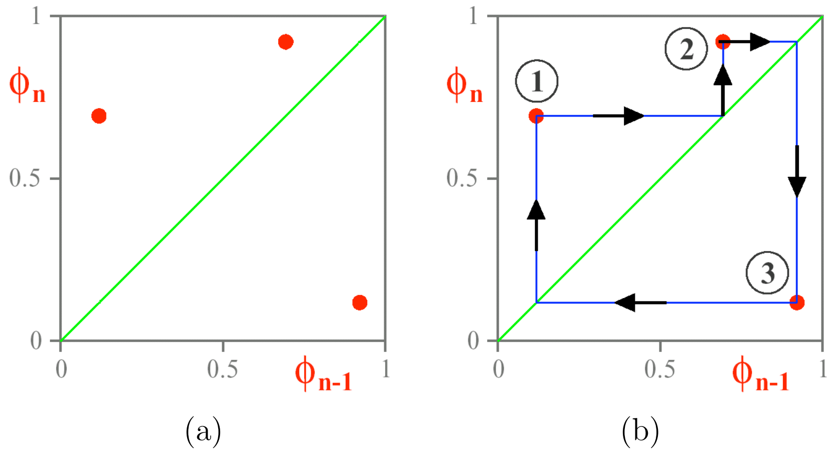


Figure 6.2: (a) Time-1 return plot and (b) Lameray diagrams for rule 62.

6.2.3 Final remarks

The time-1 characteristic function $\chi^1_{\mathcal{N}}$ is a complete representation of the local rule $\mathcal{N}$, because it contains the information needed to derive the dynamic evolutions from any initial state. Each rule has one, and only one, time-1 characteristic function. Something different happens for higher orders: for example, the time-2 characteristic function for rule 105 is exactly the same as for rule 150, because of their alternating symmetry duality [120].

In contrast, the time-1 return plot $\phi_{1,\mathcal{N}}$ is *not* unique: a local rule $\mathcal{N}$ has several of such plots (one for 'orbit', as we will see in next section) and a single plot can be shared by different rules (in terms of attractors, it means that different rules can have the same 'orbit'). What *is* peculiar to a rule is the union of all its time-1 return plots, which is also equivalent to the time-1 characteristic function. Consequently, the points of *any* time-1 return plot $\phi_{1,\mathcal{N}}$ of the local rule $\mathcal{N}$ are a subset of the time-1 characteristic function $\chi^1_{\mathcal{N}}$, and more in general the points of *any* time-τ return plot $\phi^\tau_{\mathcal{N}}$ of the local rule $\mathcal{N}$ are a subset of the time-τ characteristic function $\chi^\tau_{\mathcal{N}}$.

6.3 Analyzing CA rules through their orbits

Remarkable properties of Cellular Automata can be discovered by analyzing exhaustively the evolution of all 2^L binary bit strings with finite length L [121]. Intuitively, evolving any initial state $\mathbf{x^0}$ under any local rule $\mathcal{N}$ we will find a bit string $\mathbf{x^{T_\delta}}$ (where $\mathbf{x^{T_\delta}}$ denotes the evolution of $\mathbf{x^0}$ after T_δ iterations) that must eventually repeat itself with a period $\mathrm{T}_\Lambda \leq 2^L$. This concept can be formally defined as follows:

Definition 6.1 (Transient regime and transient duration) *Given any initial configuration $\mathbf{x^0}$ and any local rule $\mathcal{N}$, let T_δ be the smallest non-negative integer such that*

$$\mathbf{x^{T_\delta+T_\Lambda}} = \mathbf{x^{T_\delta}}$$

Since $\mathbf{x^t}$, for $t = 0, 1, \ldots, T_\delta - 1$, will never recur again for all $t \geq T_\delta$, the set whose members are the first T_δ-1 iterations of $\mathbf{x^0}$ is called transient regime originating from the initial state $\mathbf{x^0}$; the time T_δ-1 is called transient duration.

When the transient dies away, we enter a so-called *attractor*, defined as

Definition 6.2 (Period-T_Λ attractor Λ) *If T_δ >1, then the set whose members are the T_Λ consecutive (under rule $\mathcal{N}$) configurations $\mathbf{x^{T_\delta}} \ldots \boldsymbol{x}^{T_\delta+T_\Lambda-1}$ is called a period-T_Λ attractor Λ of the local rule $\mathcal{N}$ originating from the initial configuration $\mathbf{x^0}$.*

Obviously, if T_δ=0, then $\mathbf{x^0}$ is an element of the attractor. It is also important to characterize the collection of initial states evolving into configurations belonging to an attractor, through the following definition.

Definition 6.3 (Basin of attraction of Λ) *The set $\mathcal{B}_\Lambda$ of all initial states $\mathbf{x^0}$ that tend to the attractor Λ is called the basin of attraction of Λ.*

In any basin of attraction there must be at least a bit string with no predecessor, which follows under the category of *Gardens of Eden* [122]:

Definition 6.4 (Garden of Eden) *Given a rule $\mathcal{N}$, a bit string $\mathbf{x^n}$ is said to be a Garden of Eden if it has no predecessors, or in other words if there is no other bit string generating $\mathbf{x^n}$ under the rule $\mathcal{N}$:*

$$\mathbf{x^n} \textit{ is a Garden of Eden } \Leftrightarrow \nexists\, \mathbf{x^{n-1}} \in \Sigma : \ \mathbf{x^n} = T_{\mathcal{N}}(\mathbf{x^{n-1}})$$

This last definition gives rise to a special kind of orbits with no basin of attraction dubbed *Isles of Eden*:

Definition 6.5 (Isle of Eden) *An orbit in which there are no Gardens of Eden is called Isle of Eden.*

An example of attractor and Isle of Eden for rule 22 is given in Fig. 6.3. Both orbits are so-called Bernoulli: by definition, given any string belonging to a Bernoulli σ_τ-shift orbit, after τ iterations we find the same string shifted σ positions towards left, if σ is positive.

6.4 Classification of CA rules

6.4.1 Wolfram's classification

Cellular Automata local rules can be grouped according to many criteria, and several different classifications have been presented so far [123] [124] [125] [126] [127] [128] [129] [130]. Probably, the most famous is due to Wolfram, who proposed to classify CA local rules into four classes (labeled from 'W1' to 'W4'), depending on the evolution of the system from a random initial state [114]:

- **W1**: evolution leads to a homogeneous state;
- **W2**: evolution leads to a set of separated simple stable or periodic structures;
- **W3**: evolution leads to a chaotic pattern;
- **W4**: evolution leads to complex localized structures, sometimes long-lived.

On the one hand, this classification makes clear that only certain rules can perform computation, and then they are somehow 'interesting'; on the other

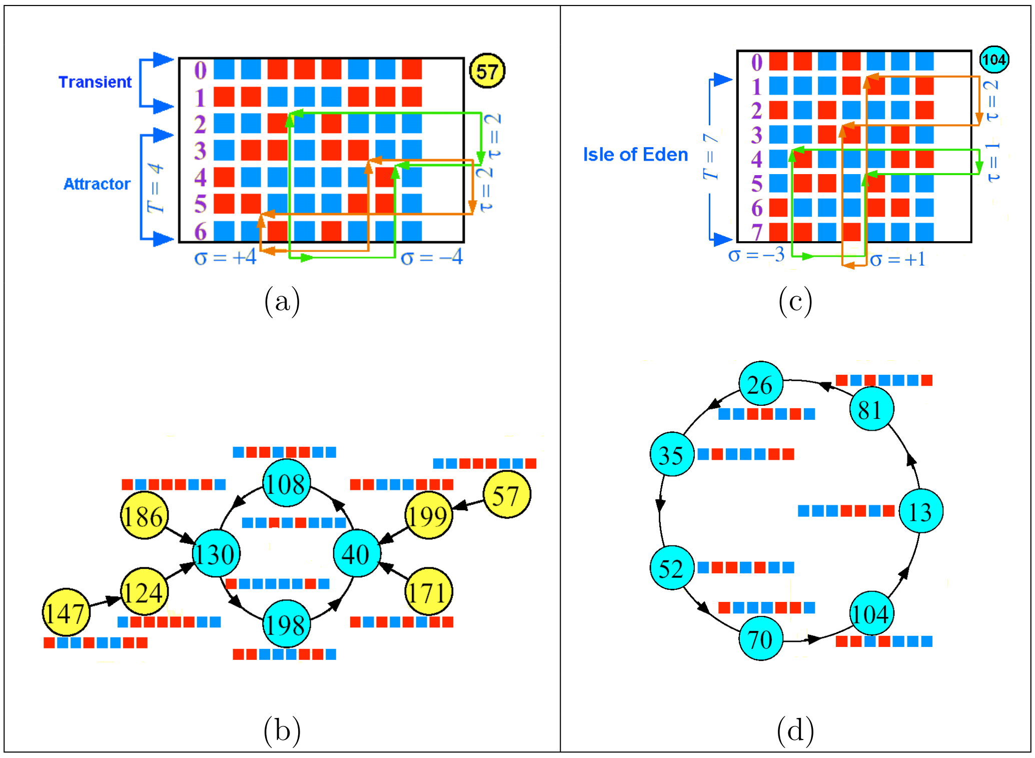


Figure 6.3: Example of attractor (a) and (b) with transient period T_δ=2 and period T_Λ=4, and Isle of Eden with period T_Λ=7 for rule 22 and $L = 9$. The parameters σ and τ refer to the fact that both are Bernoulli σ_τ-shift orbits, as it will be explained in Sec. 7.3.

hand, it has received a lot of criticism for being based on empirical criteria [131]. Moreover, this method is not able to classify all rules of CA with a number of states greater than 2 and/or radius of influence greater than 1 ([28], pag. 241), and, in general, the class membership of a given rule is undecidable [132].

6.4.2 Chua's classification

An alternative classification scheme composed by six different groups (labeled from 'C1' to 'C6') was proposed by Chua in [120]. In this case the feature used to discriminate rules is the *robust* behavior of attractors found by using random bit strings:

- **C1**: local rules exhibiting a robust period-1 steady-state behavior, corresponding to fixed points of the time-1 characteristic function $\chi^1_\mathcal{N}$ of local rule $\mathcal{N}$;

- **C2**: local rules exhibiting a robust period-2 steady-state behavior, corresponding to fixed points of the time-2 characteristic function $\chi^2_{\mathcal{N}}$ of local rule $\mathcal{N}$;

- **C3**: local rules exhibiting a robust period-3 steady-state behavior, corresponding to fixed points of the time-3 characteristic function $\chi^3_{\mathcal{N}}$ of local rule $\mathcal{N}$;

- **C4**: local rules exhibiting a robust σ_τ-shift steady-state behavior corresponding to a period-T attractor or Isle of Eden, where $T \leq \sigma L$;

- **C5**: bilateral local rules exhibiting a robust σ_τ-shift steady-state behavior in which the parameters σ and τ depend on the initial point or the length L;

- **C6**: non-bilateral local rules exhibiting a robust σ_τ-shift steady-state behavior, in which the parameters σ and τ depend on the initial point or the length L.

A complete classification of the 88 globally independent CA local rules into these six classes is given in Table 6.1.
Roughly speaking, there are three macro groups. The first one (44 rules) is composed by local rules belonging to **C1**, **C2**, and **C3**, having non-Bernoulli orbits (attractors and Isles of Eden) with very short period; surprisingly, the maximum *robust* period is 3, which is verified only by rule 62. The second macro group coincides with **C4**, including 30 Bernoulli-σ_τ rules in which the values assumed by the two parameters are either fixed (like in rule 240, for which $\sigma = 1$ and $\tau = 1$) or in general do not depend on the length L and the initial state, modulo a finite number of basins of attraction (like in rule 9 [133]). The last macro group (18 rules) includes **C5** and **C6**, whose rules have attractors characterized by complex values of σ and τ, unrelated with L.
It is import to remark that Chua's classification refers exclusively to *robust* properties; then, rules may have *non-robust* attractors of a different kind of the *robust* ones. For example, rule 168 has *robust* period-1 orbits, but it also shows a chaotic behavior for specific initial conditions [134].

Table 6.1: Classification of the 88 globally independent CA local rules according to Chua.

26 Period-1 rules					
0	**4**	**8**	**12**	**13**	**32**
28	**40**	**44**	**72**	**76**	**77**
78	**94**	**104**	**128**	**132**	**136**
140	**160**	**164**	**168**	**172**	**200**
204	**232**				

13 Period-2 rules			
1	**5**	**19**	**23**
28	**29**	**33**	**37**
50	**51**	**108**	**156**
178			

30 Bernoulli σ_τ-rules					
2	**3**	**6**	**7**	**9**	**10**
11	**14**	**15**	**24**	**25**	**27**
34	**35**	**38**	**42**	**43**	**46**
56	**57**	**58**	**74**	**130**	**134**
138	**142**	**152**	**162**	**170**	**184**

1 Period-3 rule			
62			

10 Complex Bernoulli-shift rules					
18	**22**	**54**	**73**	**90**	**105**
122	**126**	**146**	**150**		

8 Hyper Bernoulli-shift			
26	**30**	**41**	**45**
60	**106**	**110**	**154**

6.4.3 Relationship between Wolfram's and Chua's classifications

These two classifications are related one to the other: rules belonging to **W1** have a global attractor and hence they are a subgroup of **C1**; rules belonging to **W2** can in principle be in any of the first four Chua's groups; finally, rules belonging to **W3** and **W4**, can be either in **C5** or in **C6**, depending on their computational capability. Therefore, we can summarize these results as follows:

$$\mathbf{W1} \subset \mathbf{C1}; \quad \mathbf{W2} \equiv ((\mathbf{C1} \setminus \mathbf{W1}) \cup \mathbf{C2} \cup \mathbf{C3} \cup \mathbf{C4}); \quad (\mathbf{W3} \cup \mathbf{W4}) \equiv (\mathbf{C5} \cup \mathbf{C6})$$

Note that this analogy may not hold for two-dimensional CA, for which a systematic study from nonlinear dynamics perspective does not exist yet.

6.4.4 Additive rules

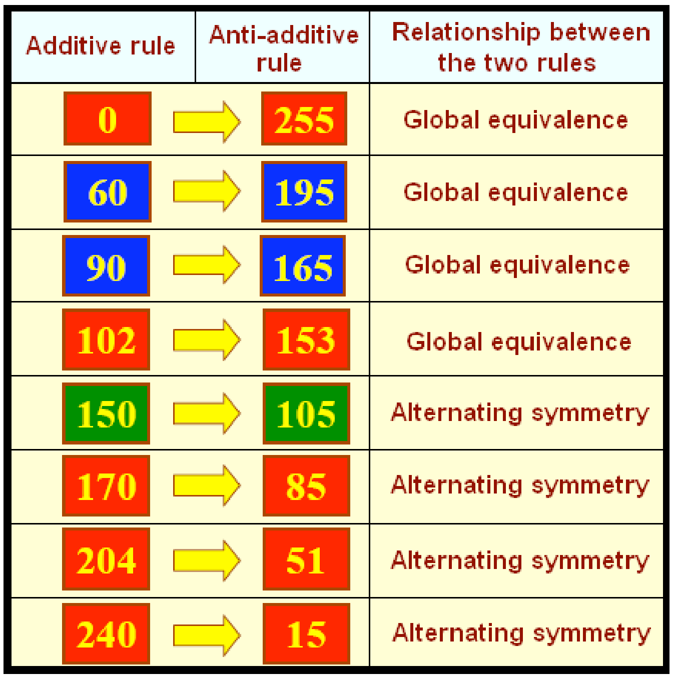
Among the other classifications that can be done among CA local rules, we want to mention that into *additive* and *nonadditive* rules. Additive rules can be described through the expression

$$\mathbf{x_i^{t+1}} = \mathrm{T}_{\mathcal{N}}(\mathbf{x_{i-1}^{t} x_i^{t} x_{i+1}^{t}}) \tag{6.8}$$

Since this equation contains three free parameters, there are $2^3 = 8$ distinct additive rules, which can be represented through simple Boolean functions as follows

(abc)=(000):	$\mathbf{x_i^{t+1}} = 0 \pmod 2$	$\Rightarrow$ Rule 0
(abc)=(001):	$\mathbf{x_i^{t+1}} = \mathbf{x_{i+1}^{t}} \pmod 2$	$\Rightarrow$ Rule 170
(abc)=(010):	$\mathbf{x_i^{t+1}} = \mathbf{x_i^{t}} \pmod 2$	$\Rightarrow$ Rule 204
(abc)=(011):	$\mathbf{x_i^{t+1}} = \mathbf{x_i^{t}} + \mathbf{x_{i+1}^{t}} \pmod 2$	$\Rightarrow$ Rule 102
(abc)=(100):	$\mathbf{x_i^{t+1}} = \mathbf{x_{i-1}^{t}} \pmod 2$	$\Rightarrow$ Rule 240
(abc)=(101):	$\mathbf{x_i^{t+1}} = \mathbf{x_{i-1}^{t}} + \mathbf{x_{i+1}^{t}} \pmod 2$	$\Rightarrow$ Rule 90
(abc)=(110):	$\mathbf{x_i^{t+1}} = \mathbf{x_{i-1}^{t}} + \mathbf{x_i^{t}} \pmod 2$	$\Rightarrow$ Rule 60
(abc)=(111):	$\mathbf{x_i^{t+1}} = \mathbf{x_{i-1}^{t}} + \mathbf{x_i^{t}} + \mathbf{x_{i+1}^{t}} \pmod 2$	$\Rightarrow$ Rule 150

Each additive rule $\mathcal{N}$ has a complementary anti-additive rule $\mathcal{N}^c = 255 - \mathcal{N}$. Table 6.4 shows that 4 additive rules are globally equivalent to their corresponding anti-additive rules, whereas in the remaining four cases additive and anti-additive rules exhibit an alternating symmetry duality [120].
There are only 9 globally equivalent additive rules: 0, 15, 51, 60, 90, 105, 150, 170, 204. Five of them - namely 0, 15, 51, 170, 204 - have complexity index $\kappa = 1$, and their behavior can be easily analyzed; for this reason we dub them *trivial* additive rules. In contrast, the remaining four rules - 60, 90, 105, and 150 - have complexity index $\kappa = 2$ (rules 60 and 90) or $\kappa = 3$ (rules 105 and 150). According to Chua's classification, rule 60 belongs to group 6 (hyper Bernoulli-shift rules) and rules 90, 105, 150 belong to group 5 (complex Bernoulli-shift rules); therefore, they exhibit complex behaviors, and we dub them *non-trivial* additive rules.



Additive rule	Anti-additive rule	Relationship between the two rules
0	255	Global equivalence
60	195	Global equivalence
90	165	Global equivalence
102	153	Global equivalence
150	105	Alternating symmetry
170	85	Alternating symmetry
204	51	Alternating symmetry
240	15	Alternating symmetry

Figure 6.4: Relationship between additive and anti-additive rules.

Chapter 7

New results on Cellular Automata

Thanks to the nonlinear dynamics perspective of Cellular Automata, it has been possible to find many new results, four of which are illustrated in this chapter. We start from quasi-ergodicity, a concept defined empirically but that gives a further reason to justify the CA local rules classification proposed by Chua. Then, we describe how and why fractals emerge naturally in time-1 characteristic functions of any local rule, making a further classification of rules according to the number of fractal patterns they present. Third, we give some results on Bernoulli orbits, which have important practical consequences since they allow us to reduce sensibly the number of cases that have to be simulated. Finally, we give a quantitative explanation of the scale-free property for additive CA rules, summarizing the results in a theorem and illustrating them through several examples.

7.1 Quasi-ergodicity

The time-1 return map and the time-1 characteristic function are two different kinds of visual representation of a CA rule: the first one illustrates the evolution from a specific initial bit string as a Lamerey (cobweb) diagram for all times; the second one shows the output bit string corresponding to *any* possible input for one iteration. In general, a local rule has a unique time-1 characteristic function but many time-1 return maps, one for each initial condition.
Therefore, a question arises: can the time-1 return map and the time-1 characteristic function coincide? In other words, is it possible to extract information about the temporal behavior which is the time-1 return map of a

Cellular Automaton by looking at its spatial representation, like the time-1 characteristic function? The correspondence between the temporal and the spatial behaviors of dynamical systems is a well-known concept dubbed 'ergodicity', and it is fundamental for a number of branches of mathematics and physics. Since a thorough treatment of ergodicity would go beyond the purpose of this dissertation, we will opt for a pedagogically more transparent approach by introducing the empirical concept 'quasi-ergodicity' to stress its difference with 'ergodicity', which would require an in-depth measure-theoretic analysis. Note that is this context we use the expression 'quasi-ergodicity' in a different, albeit related, sense with respect to what usually done in statistical physics.
Remarkably, 'quasi-ergodicity' captures a peculiarity unique among the complex and hyper Bernoulli rules, thereby adding a further raison d'être for the classification proposed by Chua, besides those based on the period or the Bernoulli σ_τ-shift of the attractors defined in [120].

By visual inspection, we noticed that for certain rules the time-1 return maps corresponding to different initial conditions are indistinguishable, and they tend to be very similar to the time-1 characteristic function too. As we mentioned before, this behavior is somehow related to ergodicity but, in order to emphasize that this phenomenon has been observed empirically, we preferred to introduce the property of quasi-ergodicity via the following definition.

Definition 7.1 (Quasi-ergodicity) *A rule is said to be quasi-ergodic iff given an arbitrary point $\mathcal{P}$ belonging to an attractor $\Lambda_{\mathcal{P}}$, it is possible to find a point $\mathcal{Q}$ arbitrarily close to it which belongs to another attractor $\Lambda_{\mathcal{Q}}$.*

A graphical illustration of the quasi-ergodic property is given in Fig. 7.1 for a generic rule $\mathcal{N}$. Figure 7.2 shows one of the consequences of quasi-ergodicity of rule 110: since the time-1 return maps corresponding to the 3 different initial conditions cling arbitrarily close to each other, the Lamerey (cobweb) diagrams (Fig. 7.2 (a), (b) and (c)) obtained from their superposition in Fig. 7.2 (d) are visually indistinguishable.
We can not overemphasize that Definition 7.1 is only an empirical definition for the new phenomenon to be described in this section. In particular, we have found that only the complex and hyper Bernoulli rules, corresponding to the 5th and 6th group of Chua's classification [120], are quasi-ergodic. This observation is remarkable because it introduces a particular feature possessed by only rules belonging to these two groups, within the more general class of Bernoulli-shift rules. So far, our distinction between the Bernoulli σ_τ-shift rules from group 4 and the complex and hyper Bernoulli-shift rules from

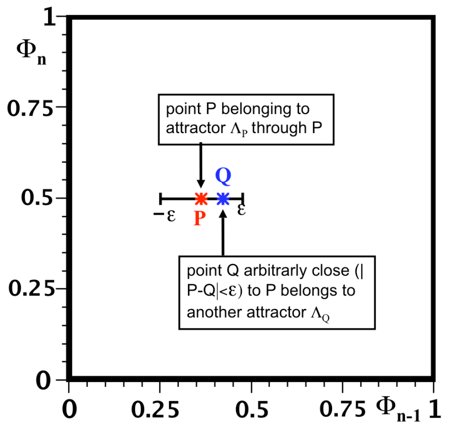

point P belonging to attractor Λ_P through P
Q
$-\varepsilon$
P
ε
point Q arbitrarly close (|P-Q|<ε) to P belongs to another attractor Λ_Q
Φ_n
Φ_{n-1}
0
0.25
0.5
0.75
1

Figure 7.1: Quasi-ergodic property

group 5 and 6 is that the two Bernoulli parameters σ and τ from group 4 do not depend on the initial point, or the length L.

All 18 complex and hyper Bernoulli rules have Lamerey (cobweb) diagrams similar to those in Fig. 7.2, with the exception of rule 73, because in this case different initial conditions generate different time-1 return maps. Nevertheless, we noticed that rule 73 has only two kinds of possible time-1 return maps corresponding to the two sets of mutually exclusive initial conditions (except, as usual, a finite number of isolated points) exhibited in Fig. 7.3. Moreover, all initial conditions belonging to the same set give indistinguishable time-1 return maps. Hence, instead of having quasi-ergodicity all over the axis, there are two regions, each exhibiting quasi-ergodicity. Therefore, we say that rule 73 is *weakly quasi-ergodic*, according to the following definition.

Definition 7.2 (Weak Quasi-ergodicity) *A rule is said to be weakly quasi-ergodic iff it is quasi-ergodic in a finite number of disjoint regions, whose union gives the whole axis.*

Among the 256 local rules, 73 and its global topologically-equivalent rule 109 are the only ones to exhibit weak quasi-ergodicity. The property of weak quasi-ergodicity of rule is evident from Fig. 7.3, in which the two different time-1 return maps are depicted in Fig. 7.3 (a) and Fig. 7.3 (b) and then their union in Fig. 7.3 (c) is compared with the time-1 characteristic function χ^1_{73} in Fig. 7.3 (d).

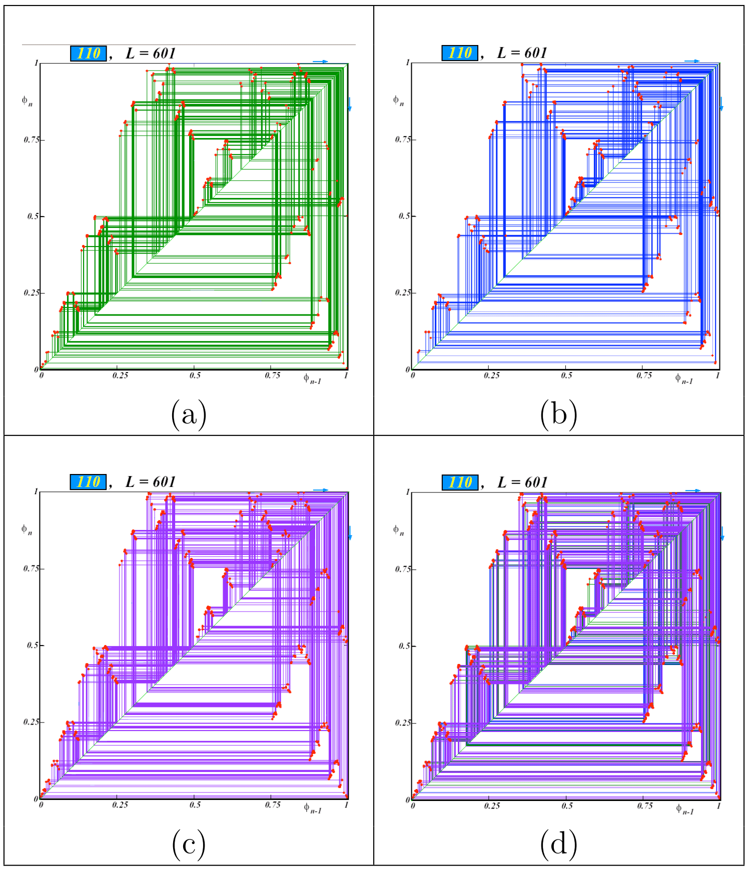


Figure 7.2: For rule 110, time-1 return maps corresponding to the 3 different initial conditions (a), (b), and (c) are visually indistinguishable, as confirmed by their superposition in (d). Then, rule 110 is quasi-ergodic.

Finally, we can identify a further type of rules within the quasi-ergodic ones thanks to the following observation. While it is true that the time-1 return map of most quasi-ergodic rules tends to cover *almost everywhere* the unit interval $\phi_{n-1} \in [0, 1]$, sometimes relatively large compact regions are excluded; in general, this behavior is observable by increasing the length of the bit strings. Nonetheless, for some rules we cannot see any gap in a generic time-1 return map independently from the length L of the bit strings. These rules namely 30, 45, 60, 90, 105, 106, 150, 154 will henceforth be called *strongly quasi-ergodic.*

Definition 7.3 (Strong Quasi-ergodicity) *A rule is said to be weakly quasi-ergodic, iff it is quasi-ergodic in a finite number of disjoint regions, whose union gives the whole axis.*

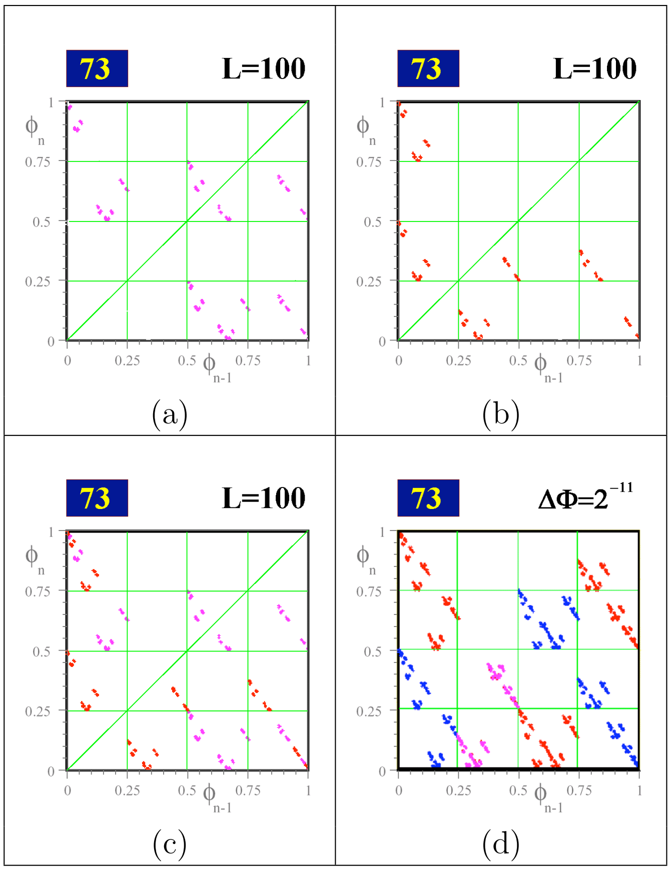


Figure 7.3: For rule 73, there are two different time-1 return maps (a) and (b) corresponding to different sets of initial conditions, and their superposition (c) coincides (except for a finite number of points) with the time-1 characteristic function (d). Then, rule 73 is weakly quasi-ergodic.

It is also possible to prove that strongly quasi-ergodic rules, along with rule 15 and 170, are indeed ergodic, according to a formal mathematical definition [135] [136].

7.2 Fractality in CA

Fractals arise naturally in the time-1 characteristic functions of one-dimensional Cellular Automata, and here we give thorough results about this phenomenon, providing also an analytical formula for finding the fractal patterns. In order to specify explicitly the number of bits contained in a generic string **x**, we introduce the notation

$$\mathbf{x_{I+1}} = (x_0 \cdots x_{k-1} \vdots \; \vdots x_I) \tag{7.1}$$

to indicate the set of bit strings composed of $I+1$ elements in which the first k elements and the last one are known. For example, $\mathbf{x_6} = (01\vdots \; \vdots 1)$ is equivalent to the set $\{(010001), (010011), (010101), (010111), (011001), (011011), (011101), (011111)\}$. Since any bit string $\mathbf{x}$ can be uniquely associated with a real number $\phi(\mathbf{x}) \in [0,1)$, the set of bit strings defined by Eq. 7.1 corresponds to a subset of points of the unit interval $[0,1]$, with the formula

$$\Phi_{I+1}(\mathbf{x_{I+1}}) = \phi_{I+1}(x_0 \cdots x_{k-1} \vdots \; \vdots x_I) = \sum_{i=0}^{I} x_i \cdot 2^{-(i+1)} \tag{7.2}$$

Similarly, the time-1 characteristic function for the local rule $\mathcal{N}$ corresponding to the set of bit strings $\phi_{I+1}(\mathbf{x_{I+1}})$ is defined by

$$\chi^1_{\mathcal{N},I+1}(\mathbf{x_{I+1}}) = \chi^1_{\mathcal{N},I+1}(x_0 \cdots x_{k-1} \vdots \; \vdots x_I) = \sum_{i=0}^{I} T_{\mathcal{N}}(x_{i-1}x_i x_{i+1}) \cdot 2^{-(i+1)} \tag{7.3}$$

where $T_{\mathcal{N}}(x_{i-1}x_i x_{i+1})$ denotes the application of the local rule $\mathcal{N}$ to the triplet $(x_{i-1}x_i x_{i+1})$. To avoid clutter, we will henceforth delete the commas separating bits, and abbreviate $\chi^1_{\mathcal{N},I+1}(\mathbf{x_{I+1}})$ by $\chi_{,I+1}(\mathbf{x_{I+1}})$.

7.2.1 All time-1 characteristics are fractal

The main result of this section is the following

Theorem 7.1 (Fractality of the time-1 characteristic function $\chi^1_{\mathcal{N}}$) *The time-1 characteristic function $\chi^1_{\mathcal{N}}$ of any rule $\mathcal{N}$ exhibits a fractal behavior in any subinterval of $\phi \in [0,1)$.*

Proof

Given an arbitrary natural number $I > 1$, we define the two bit strings $\mathbf{x^\alpha_{I+1}} = (0\vdots \; \vdots 0)$ and $\mathbf{x^\beta_{I+1}} = (1\vdots \; \vdots 0)$. According to the notation introduced previously, $\mathbf{x^\alpha_{I+1}}$ is the set of the bit strings composed by $I+1$ elements with $x_0 = 0$ and $x_I = 0$, whereas $\mathbf{x^\beta_{I+1}}$ is the set of the bit strings composed by $I+1$ elements with $x_0 = 1$ and $x_I = 0$. Their equivalent decimal representations are $\phi^\alpha = \phi_{I+1}(0\vdots \; \vdots 0)$ and $\phi^\beta = \phi_{I+1}(1\vdots \; \vdots 0)$, and it is easy to prove that $0 \le \phi^\alpha \le \frac{1}{2}$ and $\frac{1}{2} \le \phi^\beta \le 1$. Since $\Phi^\alpha \cup \Phi^\beta = [0,1)$ as $I \to \infty$ we can

divide the time-1 characteristic function $\chi^1_{\mathcal{N}}$ for a generic rule $\mathcal{N}$ into two parts, one for each interval: $\chi^\alpha = \chi_{I+1}(0\vdots\ \vdots 0)$ in correspondence to Φ^α and $\chi^\beta = \chi_{I+1}(1\vdots\ \vdots 0)$ in correspondence to Φ^β (see Fig. 7.4(a)).
An analogous procedure can be followed when $x_I = 1$, leading to the definition of the two strings $\mathbf{x}^\gamma_{\mathbf{I+1}} = (0\vdots\ \vdots 1)$ and $\mathbf{x}^\delta_{\mathbf{I+1}} = (1\vdots\ \vdots 1)$, and their equivalent decimal representations $\phi^\gamma = \phi_{I+1}(0\vdots\ \vdots 1)$ and $\phi^\delta = \phi_{I+1}(1\vdots\ \vdots 1)$, where $0 \le \phi^\gamma \le \frac{1}{2}$ and $\frac{1}{2} \le \phi^\delta \le 1$. Also in this case the time-1 characteristic function can be divided into two parts: $\chi^\gamma = \chi_{I+1}(0\vdots\ \vdots 1)$ in correspondence to Φ^γ and $\chi^\delta = \chi_{I+1}(1\vdots\ \vdots 1)$ in correspondence to Φ^δ (see Fig. 7.4(b)).
Let us consider a generic set of bit strings $\mathbf{x_{k+I+1}}$, each composed of $k+I+1$

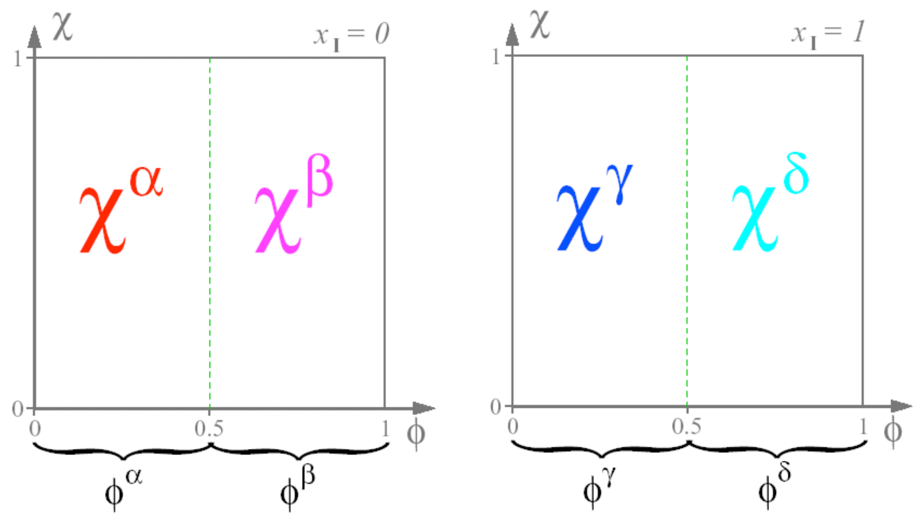


Figure 7.4: Intervals for the fractal patterns.

elements, having in common the first $k+1$ and the last element

$$\mathbf{x_{k+I+1}}\{\underbrace{\overbrace{x_0x_1\dots x_{k-1}}^{k\ bits}, \overbrace{x_k\vdots\ \vdots x_{k+1}}^{I+1\ bits}}_{k+I+1\ \text{elements}}\} \tag{7.4}$$

According to the Eq. 7.2, we can associate to the set $\mathbf{x_{k+I+1}}$ a unique interval of values in $[0, 1)$ called ϕ_{k+I+1}, in which a certain ϕ_{k+I+1} can be defined. Our purpose is to explore the behavior of the function ϕ_{k+I+1} in subintervals of ϕ_{k+I+1}, thereby showing the emergence of fractal patterns. In general, splitting ϕ_{k+I+1} into n parts corresponds to introducing $\log_2 n$ bits after x_k in the set of bit strings $\mathbf{x}_{k+I+1}$. For example, we can subdivide ϕ_{k+I+1} into four regions just by adding two extra bits called x_{left} and x_{right} after x_k in 7.4, and creating the new set of bit strings $\mathbf{x}_{k+I+3}$

$$\mathbf{x'_{k+I+3}} = (\overbrace{x_0 x_1 \ldots x_k x_{left}}^{k+2\ bits}, \overbrace{x_{right} \vdots \vdots x_{k+1}}^{I+1\ bits}) = (\overbrace{x'_0 x'_1 \ldots x'_{k+I+2}}^{k+I+3\ bits}) \tag{7.5}$$

The correspondence between $\mathbf{x_{k+I+1}}$ and $\mathbf{x'_{k+I+3}}$ can be obtained by comparing 7.4 and 7.5.

$$\begin{cases} x'_0 & = & x_0 \\ & \vdots & \\ x'_k & = & x_k \\ x'_{k+1} & = & x_{left} \\ x'_{k+2} & = & x_{right} \\ x'_{k+3} & = & x_{k+1} \\ & \vdots & \\ x'_{k+I+2} & = & x_{k+I} \end{cases} \tag{7.6}$$

The first of the four subregions corresponds to the case $(x_{left}x_{right})= (00)$, the second subregion to $(x_{left}x_{right})= (01)$, the third subregion to $(x_{left}x_{right})= (10)$, and finally the fourth subregion to $(x_{left}x_{right})= (11)$. The expression for the time-1 characteristic function $\chi_{k+I+3}(\mathbf{x'_{k+I+3}})$ can be found by using the Eq. 7.3, obtaining

$$\chi_{k+I+3}(\mathbf{x'_{k+I+3}}) = \sum_{i=0}^{k+I+2} T_{\mathcal{N}}(x'_{i-1}x'_i x'_{i+1}) \cdot 2^{-(i+1)} \tag{7.7}$$

$$= \sum_{i=0}^{k+1} T_{\mathcal{N}}(x'_{i-1}x'_i x'_{i+1}) \cdot 2^{-(i+1)} + \sum_{i=k+2}^{k+I+2} T_{\mathcal{N}}(x'_{i-1}x'_i x'_{i+1}) \cdot 2^{-(i+1)}$$

Let us analyze separately the two terms of this last equation.
By using the Eq. 7.6, we find that the first term is equivalent to

$$\sum_{i=0}^{k+1} T_{\mathcal{N}}(x'_{i-1}x'_i x'_{i+1}) \cdot 2^{-(i+1)} = \frac{1}{2} T_{\mathcal{N}}(x_{k+I-1}x_0 x_1) + \sum_{i=1}^{k-1} T_{\mathcal{N}}(x_{i-1}x_i x_{i+1}) \cdot 2^{-(i+1)}$$
$$+ \frac{1}{2^{k+1}} T_{\mathcal{N}}(x_{k-1}x_k x_{left}) + \frac{1}{2^{k+2}} T_{\mathcal{N}}(x_k x_{left} x_{right}) \tag{7.8}$$

Since for any local rule $\mathcal{N}$ the value $T_{\mathcal{N}}(x'_{i-1}x'_i x'_{i+1})$ is either 1 or -1, from Eq. 7.8 it follows that $\sum_{i=0}^{k+1} T_{\mathcal{N}}(x'_{i-1}x'_i x'_{i+1}) \cdot 2^{-(i+1)}$ is a rational number, with

$$0 \le \sum_{i=0}^{k+1} T_{\mathcal{N}}(x'_{i-1}x'_i x'_{i+1}) \cdot 2^{-(i+1)} \le \sum_{j=1}^{k+2} < 1 \tag{7.9}$$

The second term of the formula 7.7 can be written as

$$\sum_{i=k+2}^{k+I+2} T_{\mathcal{N}}(x'_{i-1}x'_i x'_{i+1}) \cdot 2^{-(i+1)} = \sum_{j=0}^{I} T_{\mathcal{N}}(x'_{j+k+1}x'_{j+k+2}x'_{j+k+3}) \cdot 2^{-(j+k+3)} = \tag{7.10}$$

$$\frac{1}{2^{k+2}} \sum_{j=0}^{I} T_{\mathcal{N}}(x'_{j+k+1}x'_{j+k+2}x'_{j+k+3}) \cdot 2^{-(j+1)} =$$

$$= \frac{1}{2^{k+2}} \left(\frac{1}{2} T_{\mathcal{N}}(x_{left}x_{right}x_{k+1}) + \frac{1}{4} T_{\mathcal{N}}(x_{right}x_{k+1}x_{k+2}) + \right.$$

$$\left. \sum_{j=2}^{i-1} T_{\mathcal{N}}(x_{j+k-1}x_{j+k}x_{j+k+1}) \cdot 2^{-(j+1)} + \frac{1}{2^{I+1}} T_{\mathcal{N}}(x_{k+I-1}x_{k+I}x_{left}) \right)$$

$$\cong \frac{1}{2^{k+2}} \chi_{I+1}(x_{right} \vdots \vdots x_{left})$$

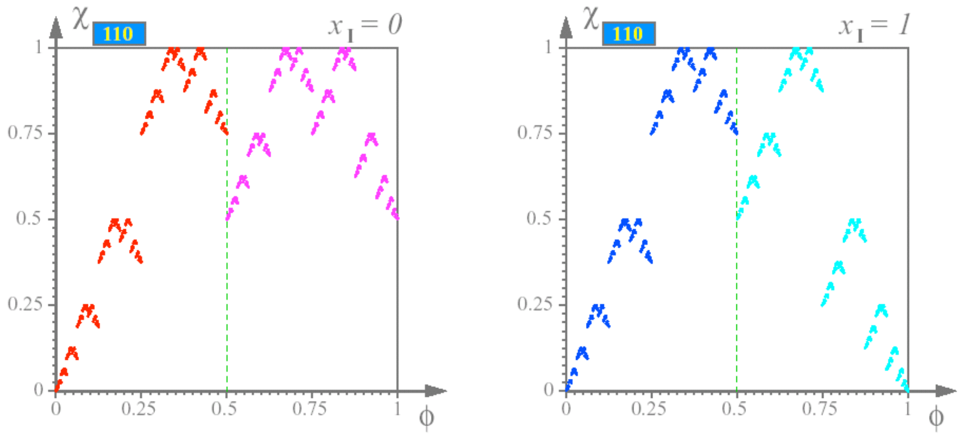
because the actual outcome of the term $\frac{1}{2^{I+1}} T_{\mathcal{N}}(x_{k+I-2}x_{k+I-1}x_{left})$ is negligible for $I \gg 1$, and $(x_{k+1} \cdots x_{k+I-1})$ are arbitrary bits, as stated in Eq. 7.4. Therefore, using Eqs. 7.8 and 7.10, we can rewrite the expression 7.7 as follows:

$$\chi_{k+I+3} = \underbrace{\sum_{i=0}^{k+1} T_{\mathcal{N}}(x'_{i-1}x'_i x'_{i+1}) \cdot 2^{-(i+1)} +}_{\text{vertical shift}} \underbrace{\frac{1}{2^{k+2}}}_{\text{scale factor}} \chi_{I+1}(x_{right} \vdots \vdots x_{left}) \tag{7.11}$$

This means that in each subregion there is a scaled and shifted copy of $\chi_{I+1}(x_{right} \vdots \vdots x_{left})$. In particular, in the first subregion $\chi_{I+1}(x_{right} \vdots \vdots x_{left}) = \chi_{I+1}(0 \cdots 0) = \chi^{\alpha}$, in the second subregion $\chi_{I+1}(x_{right} \vdots \vdots x_{left}) = \chi_{I+1}(1 \cdots 0) = \chi^{\beta}$, in the third subregion $\chi_{I+1}(x_{right} \vdots \vdots x_{left}) = \chi_{I+1}(0 \cdots 1) = \chi^{\gamma}$, and in the fourth subregion$\chi_{I+1}(x_{right} \vdots \vdots x_{left}) = \chi_{I+1}(1 \cdots 1) = \chi^{\delta}$. Since k and I are arbitrary, it follows that the time-1 characteristic function of a generic rule $\mathcal{N}$ is composed of the four fractal patterns χ^{α}, χ^{β}, χ^{γ}, and χ^{δ} (in this order) in any subinterval of $\phi \in [0, 1)$.

As an example, let us consider rule 110: its fractal patterns for the case $x_I = 0$ are shown in Fig. 7.5(a), where $\chi^{\alpha} = \chi_{I+1}(0 \cdots 0)$ is plotted in red and

$\chi^\beta = \chi_{I+1}(1\cdots 0)$ is in purple; the fractal patterns for the case $x_I = 1$ are in Fig. 7.5(b), where $\chi^\gamma = \chi_{I+1}(0\cdots 1)$ is plotted in blue and $\chi^\delta = \chi_{I+1}(1\cdots 1)$ is in cyan.



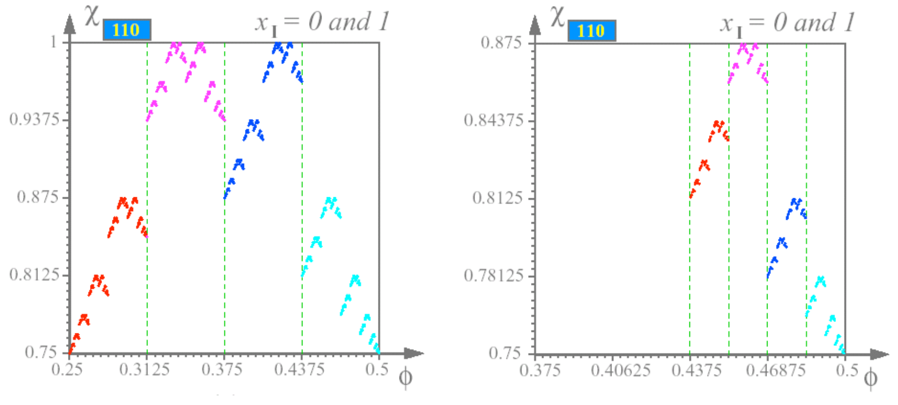
(a) (b)

Figure 7.5: Intervals for the fractal patterns of rule 110.

According to theorem 7.1, these four patterns appear naturally in any subinterval of $\phi \in [0,1)$. For instance, let us consider $\phi \in [0.25, 0.5)$, which corresponds to the set of bit strings $\mathbf{x_{I+2}} = (01\vdots\,\vdots 0)$; the restriction of χ to $\phi \in [0.25, 0.5)$ is depicted in Fig. 7.6(a). Let us divide this interval into four parts by adding two additional bits: the first subregion corresponds to the case $\mathbf{x_{I+4}} = (0100\vdots\,\vdots 0)$ and $0.25 \le \phi_{I+4} < 0.3125$, the second subregion to $\mathbf{x_{I+4}} = (0101\vdots\,\vdots 0)$ and $0.3125 \le \phi_{I+4} < 0.375$, the third subregion to $\mathbf{x_{I+4}} = (0110\vdots\,\vdots 0)$ and $0.375 \le \phi_{I+4} < 0.4375$, and the fourth subregion to $\mathbf{x_{I+4}} = (0111\vdots\,\vdots 0)$ and $0.4375 \le \phi_{I+4} < 0.5$. The analytical expressions for $\chi_{I+4}(\mathbf{x_{I+4}})$ in the four subregions can be derived from the Eqs. 7.8 and 7.11, and the results are illustrated in Fig. 7.6(a). In particular

$$\begin{aligned}
\chi_{I+4}(0100\cdots 0) = {} & \tfrac{1}{2}T_{110}(001) + \tfrac{1}{4}T_{110}(101) + \tfrac{1}{8}T_{110}(100) + \tfrac{1}{8}\chi_{I+8}(0\cdots 0) \\
& \tfrac{1}{2} + \tfrac{1}{4} + \tfrac{1}{8}\chi^\alpha = \tfrac{3}{4} + \tfrac{1}{8}\chi^\alpha \\
\chi_{I+4}(0101\cdots 0) = {} & \tfrac{1}{2}T_{110}(001) + \tfrac{1}{4}T_{110}(010) + \tfrac{1}{8}T_{110}(101) + \tfrac{1}{8}\chi_{I+8}(1\cdots 0) \\
& \tfrac{1}{2} + \tfrac{1}{4} + \tfrac{1}{8} + \tfrac{1}{8}\chi^\beta = \tfrac{7}{8} + \tfrac{1}{8}\chi^\beta \\
\chi_{I+4}(0110\cdots 0) \simeq {} & \tfrac{1}{2}T_{110}(001) + \tfrac{1}{4}T_{110}(011) + \tfrac{1}{8}T_{110}(110) + \tfrac{1}{8}\chi_{I+8}(0\cdots 1) \\
& \tfrac{1}{2} + \tfrac{1}{4} + \tfrac{1}{8} + \tfrac{1}{8}\chi^\gamma = \tfrac{7}{8} + \tfrac{1}{8}\chi^\gamma
\end{aligned}$$

$$\begin{aligned}\chi_{I+4}(0111\cdots 0) \simeq\ & \tfrac{1}{2}T_{110}(001)+\tfrac{1}{4}T_{110}(011)+\tfrac{1}{8}T_{110}(111)+\tfrac{1}{8}\chi_{I+8}(1\cdots 1)\\ & \tfrac{1}{2}+\tfrac{1}{4}+\tfrac{1}{8}\chi^{\delta}=\tfrac{3}{4}+\tfrac{1}{8}\chi^{\delta}\end{aligned}$$



(a) (b)

Figure 7.6: Emergence of fractal patterns in arbitrary subinterval of rule 110.

As expected, the time-1 characteristic function ϕ_{I+4} in the four subregions is a scaled and shifted version of χ^{α}, χ^{β}, χ^{γ}, and χ^{δ}, respectively. In Fig. 7.6(b) a further subdivision of the horizontal axis ? is depicted, and also in this case the four fractal patterns emerge naturally.

7.2.2 From the time-1 characteristic function to the rule number

The analytical characterization of the fractality of $\chi^1_{\mathcal{N}}$ for a generic rule $\mathcal{N}$ makes it possible to obtain the rule number directly from the time-1 characteristic function diagram. Here we consider strings composed by $I+1$ bits, and their decimal representation ϕ_{I+1} defined as in Eq. 6.4. Let us differentiate between $x_I=0$ and $x_I=1$: for the first case, we divide the axis ϕ into four intervals, namely $\phi^{0,I}(00\ldots0)$, $\phi^{0,II}(01\ldots0)$, $\phi^{0,III}(10\ldots0)$, and $\phi^{0,IV}(00\ldots0)$, each corresponding to a quarter of the axis ϕ. In each of these interval, the time-1 characteristic functions can be defined by means of the formula 7.11. For example, in the first subregion

$$\chi^{0,I}=\chi_{I+1}(00\ldots0)=\frac{1}{2}T_{\mathcal{N}}(000)+\frac{1}{2}\chi^{\alpha}$$

therefore

$$\chi^{0,I}>\frac{1}{2}\Leftrightarrow T_{\mathcal{N}}(000)=1$$

This means that the value of $T_{\mathcal{N}}(000)$ is completely determined by the position of the characteristic function in the first quarter of the diagram for $x_I = 0$: if the function $\chi^{0,I}$ is below $\frac{1}{2}$ then $T_{\mathcal{N}}(000) = 0$; if it is above $\frac{1}{2}$ then $T_{\mathcal{N}}(000) = 1$.
Following an analogously procedure, we find that

$$\chi^{0,II} = \chi_{I+1}(01\ldots0) = \frac{1}{2}T_{\mathcal{N}}(001) + \frac{1}{2}\chi^{\beta} \rightarrow \chi^{0,II} > \frac{1}{2} \Leftrightarrow T_{\mathcal{N}}(001) = 1$$

$$\chi^{0,III} = \chi_{I+1}(10\ldots0) \simeq \frac{1}{2}T_{\mathcal{N}}(010) + \frac{1}{2}\chi^{\gamma} \rightarrow \chi^{0,III} > \frac{1}{2} \Leftrightarrow T_{\mathcal{N}}(010) = 1$$

$$\chi^{0,IV} = \chi_{I+1}(11\ldots0) = \frac{1}{2}T_{\mathcal{N}}(011) + \frac{1}{2}\chi^{\delta} \rightarrow \chi^{0,IV} > \frac{1}{2} \Leftrightarrow T_{\mathcal{N}}(011) = 1$$

and then the values for $T_{\mathcal{N}}(001)$, $T_{\mathcal{N}}(010)$, $T_{\mathcal{N}}(011)$ depend on the position of the time-1 characteristic function in the second, third and fourth region, respectively.
As for the case $x_I = 1$, we can also divide the axis ϕ into four equal intervals, namely $\phi^{1,I}(00\ldots1)$, $\phi^{1,II}(01\ldots1)$, $\phi^{1,III}(10\ldots1)$, and $\phi^{1,IV}(00\ldots1)$. The expression for the time-1 characteristic function in the first interval can be found through the Eq. 7.11

$$\chi^{1,I} = \chi_I(00\ldots1) \simeq \frac{1}{2}(T_{\mathcal{N}}(100) + \chi_{I-1}(0\ldots0)) = \frac{1}{2}T_{\mathcal{N}}(100) + \frac{1}{2}\chi^{\alpha}$$

therefore

$$\chi^{1,I} > \frac{1}{2} \Leftrightarrow T_{\mathcal{N}}(100) = 1$$

In this case the value of $T_{\mathcal{N}}(100)$ is completely determined by the position of the characteristic function in the first quarter of the diagram for $x_I = 1$: if the function $\chi^{1,I}$ is below $\frac{1}{2}$ then $T_{\mathcal{N}}(100) = 0$; if it is above $\frac{1}{2}$ then $T_{\mathcal{N}}(100) = 1$.
We can extend the same result to the other three subregions, obtaining

$$\chi^{1,II} = \chi_{I+1}(01\ldots1) = \frac{1}{2}T_{\mathcal{N}}(101) + \frac{1}{2}\chi^{\beta} \rightarrow \chi^{1,II} > \frac{1}{2} \Leftrightarrow T_{\mathcal{N}}(101) = 1$$

$$\chi^{1,III} = \chi_{I+1}(10\ldots0) \simeq \frac{1}{2}T_{\mathcal{N}}(110) + \frac{1}{2}\chi^{\gamma} \rightarrow \chi^{1,III} > \frac{1}{2} \Leftrightarrow T_{\mathcal{N}}(110) = 1$$

$$\chi^{1,IV} = \chi_{I+1}(11\ldots0) = \frac{1}{2}T_{\mathcal{N}}(111) + \frac{1}{2}\chi^{\delta} \rightarrow \chi^{1,IV} > \frac{1}{2} \Leftrightarrow T_{\mathcal{N}}(111) = 1$$

and then the values for $T_{\mathcal{N}}(101)$, $T_{\mathcal{N}}(110)$, $T_{\mathcal{N}}(111)$ depend on the position of the time-1 characteristic function in the second, third and fourth region, respectively.
In conclusion, the number of the local rule can be found straightforwardly by visual inspection of the time-1 characteristic function diagrams, as summarized in Fig. 7.7.

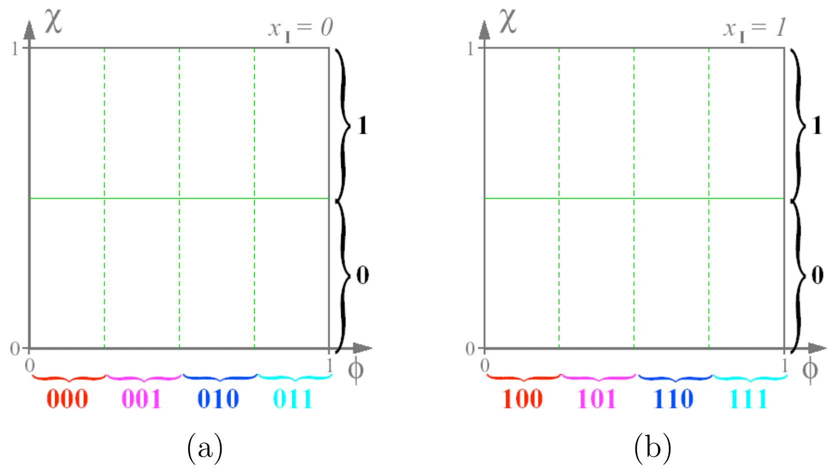


Figure 7.7: The rule number can be easily obtained by looking at the position of the time-1 characteristic function in the eight intervals.

7.3 Bernoulli orbits

Complex and hyper Bernoulli-shift rules have in general many Bernoulli orbits, either attractors of Isles of Eden, and then it is particularly important to characterize them analytically in order to avoid long and tedious simulations. Here we present a method that allows us to reconstruct the whole basin-tree diagram of a Bernoulli-shift attractor by analyzing only a fraction of all bit strings with a given L.

Examples of Bernoulli are in Fig. 6.3, whereas Fig. 7.8 shows a period T=14 Bernoulli-σ_τ orbit for rule 110, in which $\sigma = 1$, $\tau = 4$ and L=7. In general, a left shift by σ positions in the binary representation $\mathbf{x^n} = (x_0^n x_1^n \ldots x_I^n)$ is equivalent to a multiplication by 2^σ in the decimal representation n, where $n = \sum_{i=0}^{L-1} x_{L-1-i} \cdot 2^i$.

Therefore, if n_0 is a (decimal) element of a Bernoulli σ_τ-shift orbit and n_τ the result after τ iterations, $n_\tau = T_{\mathcal{N}}^\tau(n_0)$, it follows that

$$n_\tau = n_0 \cdot 2^\sigma \;\; (mod\; 2^L - 1) \qquad (7.12)$$

Furthermore, this formula can be modified to find not only n_τ, but also all the other elements of the orbit generated by n_0. A Bernoulli σ_τ-shift orbit with period T harbors exactly $lcm(\tau, T)$ different subgroups (lcm indicates the least common mutiple) each with length $\frac{T}{lcm(\tau,T)}$, henceforth called *order* of the subgroup. This means that Eq. 7.12 can be iterated $\frac{T}{lcm(\tau,T)} - 1$ times to obtain a corresponding number of strings, using at each iteration the result

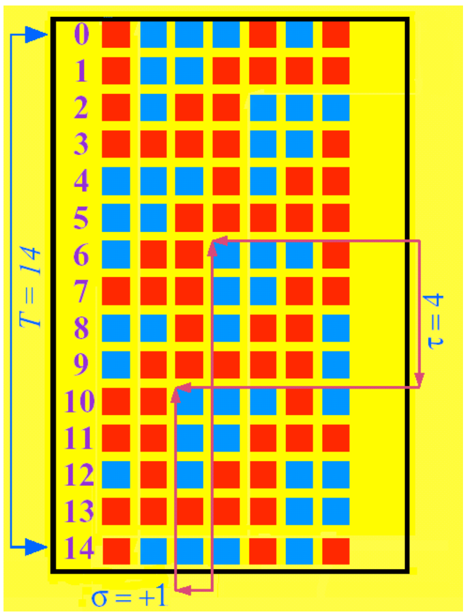


Figure 7.8: Example of a Bernoulli σ_τ-shift attractor with $\sigma = 1$, $\tau = 4$ for rule 110 and L=7.

n_τ as new value of n_0. As a consequence, Eq. 7.12 can be generalized as follows

$$n_{p\cdot\tau\ (mod\ T)} = n_0 \cdot 2^{p\cdot\sigma}\ \ (mod\ 2^L - 1), \tag{7.13}$$

where

$$0 \leq p < \frac{T}{lcm(\tau, T)}, \ \ p \in \mathbb{N}$$

where the parameter p indicates the current iteration. Obviously, iteration $\frac{T}{lcm(\tau,T)}$ would give n_0 itself.
For example, this procedure can be applied to the Bernoulli σ_τ-shift attractor with $\tau = 4$ and T=14 of rule 110 when L=7, which is shown in Fig. 7.9. From what was explained previously, there will be $lcm(\tau, T) = 2$ different subgroups, each with length $\frac{T}{lcm(\tau,T)} = 7$. Starting from an arbitrary element of period-14 Bernoulli σ_τ-shift orbit (cyan circles in Fig. 7.9), like $n_0 = 31$

and using the Eq. 7.13 we obtain the values:

$$\begin{aligned}
p = 1 &: \quad n_{1\cdot 4\,(\text{mod }14)} = n_0 \cdot 2^{1\cdot 1} \ (\text{mod } 2^7 - 1) \Rightarrow n_4 = 31 \cdot 2 \ (\text{mod } 127) = 62 \\
p = 2 &: \quad n_{2\cdot 4\,(\text{mod }14)} = n_0 \cdot 2^{2\cdot 1} \ (\text{mod } 2^7 - 1) \Rightarrow n_8 = 31 \cdot 4 \ (\text{mod } 127) = 124 \\
p = 3 &: \quad n_{3\cdot 4\,(\text{mod }14)} = n_0 \cdot 2^{3\cdot 1} \ (\text{mod } 2^7 - 1) \Rightarrow n_{12} = 31 \cdot 8 \ (\text{mod } 127) = 121 \\
p = 4 &: \quad n_{4\cdot 4\,(\text{mod }14)} = n_0 \cdot 2^{4\cdot 1} \ (\text{mod } 2^7 - 1) \Rightarrow n_2 = 31 \cdot 16 \ (\text{mod } 127) = 115 \\
p = 5 &: \quad n_{5\cdot 4\,(\text{mod }14)} = n_0 \cdot 2^{5\cdot 1} \ (\text{mod } 2^7 - 1) \Rightarrow n_6 = 31 \cdot 32 \ (\text{mod } 127) = 103 \\
p = 6 &: \quad n_{6\cdot 4\,(\text{mod }14)} = n_0 \cdot 2^{6\cdot 1} \ (\text{mod } 2^7 - 1) \Rightarrow n_{10} = 31 \cdot 64 \ (\text{mod } 127) = 79
\end{aligned}$$

Finally, for $p = \frac{T}{lcm(\tau,T)} = 7$, $n_{7\cdot 4\,(\text{mod }14)} = n_0$, as expected.

$$p = 7 : \quad n_{7\cdot 4\,(\text{mod }14)} = n_0 \cdot 2^{7\cdot 1} \ (\text{mod } 2^7 - 1) \Rightarrow n_0 = 31 \cdot 128 \ (\text{mod } 127) = 31$$

The seven elements belonging to the second subgroup, which can be iden-

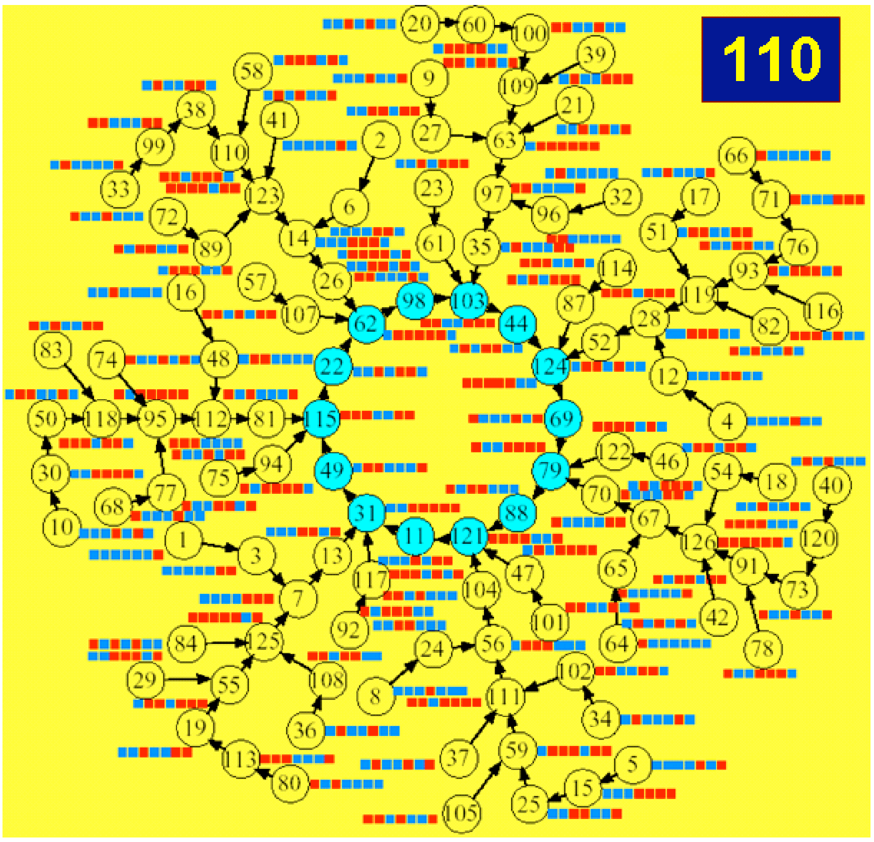


Figure 7.9: Basin-tree diagram of the Bernoulli σ_τ-shift attractor with $\sigma = 1$, $\tau = 4$ for rule 110 and L=7.

tified in the period-14 Bernoulli orbit in Fig. 7.9), can be found by using a different starting point. Since $T_{110}(31) = 49$, $n_0 = 49$ is a natural candidate; hence it follows that:

$$\begin{array}{llll}
p=1: & n_{1\cdot 4\,(\mathrm{mod}\ 14)} = n_0 \cdot 2^{1\cdot 1} \pmod{2^7-1} & \Rightarrow n_4 = 49 \cdot 2 \pmod{127} = 98 \\
p=2: & n_{2\cdot 4\,(\mathrm{mod}\ 14)} = n_0 \cdot 2^{2\cdot 1} \pmod{2^7-1} & \Rightarrow n_8 = 49 \cdot 4 \pmod{127} = 69 \\
p=3: & n_{3\cdot 4\,(\mathrm{mod}\ 14)} = n_0 \cdot 2^{3\cdot 1} \pmod{2^7-1} & \Rightarrow n_{12} = 49 \cdot 8 \pmod{127} = 11 \\
p=4: & n_{4\cdot 4\,(\mathrm{mod}\ 14)} = n_0 \cdot 2^{4\cdot 1} \pmod{2^7-1} & \Rightarrow n_2 = 49 \cdot 16 \pmod{127} = 22 \\
p=5: & n_{5\cdot 4\,(\mathrm{mod}\ 14)} = n_0 \cdot 2^{5\cdot 1} \pmod{2^7-1} & \Rightarrow n_6 = 49 \cdot 32 \pmod{127} = 44 \\
p=6: & n_{6\cdot 4\,(\mathrm{mod}\ 14)} = n_0 \cdot 2^{6\cdot 1} \pmod{2^7-1} & \Rightarrow n_{10} = 49 \cdot 64 \pmod{127} = 88 \\
p=7: & n_{7\cdot 4\,(\mathrm{mod}\ 14)} = n_0 \cdot 2^{7\cdot 1} \pmod{2^7-1} & \Rightarrow n_0 = 49 \cdot 128 \pmod{127} = 49
\end{array}$$

Also in this case, when $p = \frac{T}{lcm(\tau,T)} = 7$ we find the initial value $n_0 = 49$, as expected.

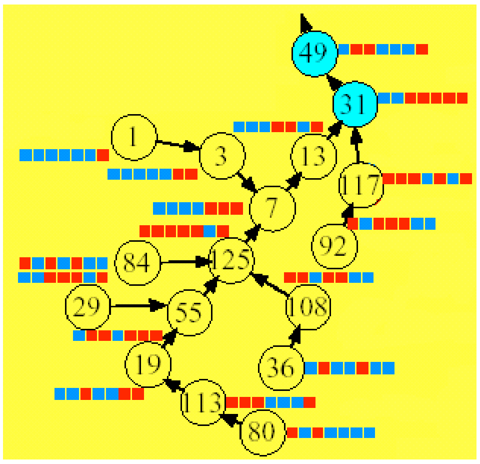


Figure 7.10: Detail of the basin-tree diagram of the Bernoulli σ_τ-shift attractor with $\sigma = 1$, $\tau = 4$ for rule 110 and L=7. These 17-bit strings allow us to reconstruct the whole basin-tree diagram, thanks to formula 7.13

Thanks to Eq. 7.13 all 14 elements of the Bernoulli $\sigma_\tau - shift$ orbit were found starting just from n_0=31 and $n_0 = 49$. Remarkably, this formula can be also used to find elements that do not belong to the orbit but to the basin tree. This is because in general

$$\mathbf{x^n} \xrightarrow{T_{\mathcal{N}}} \mathbf{x^{n+1}} \Rightarrow \sigma(\mathbf{x^n}) \xrightarrow{T_{\mathcal{N}}} \sigma(\mathbf{x^{n+1}}) \tag{7.14}$$

where $\sigma(\mathbf{x^n})$ is the string obtained by shifting σ times the bit string $\mathbf{x^n}$. For example, since $T_{110}(13) = 31$, we can expect that a shifted version of the bit string representing 31 is obtained by transforming under the rule $\mathcal{N}$ the shifted bit string representing 13.

$$p = 1: \quad n_{1\cdot 4\,(\mathrm{mod}\,14)} = n_0 \cdot 2^{1\cdot 1} \;(\mathrm{mod}\; 2^7 - 1) \Rightarrow n_4 = 13 \cdot 2 \;(\mathrm{mod}\; 127) = 26$$

and indeed

$$13 \xrightarrow{T_{110}} 31 \;\Rightarrow\; \sigma(13) = 26 \xrightarrow{T_{110}} \sigma(31) = 62$$

The values for $n_0 = 13$ are

$$\begin{array}{ll}
p = 2: & n_{2\cdot 4\,(\mathrm{mod}\,14)} = n_0 \cdot 2^{2\cdot 1} \;(\mathrm{mod}\; 2^7 - 1) \Rightarrow n_8 = 13 \cdot 4 \;(\mathrm{mod}\; 127) = 52 \\
p = 3: & n_{3\cdot 4\,(\mathrm{mod}\,14)} = n_0 \cdot 2^{3\cdot 1} \;(\mathrm{mod}\; 2^7 - 1) \Rightarrow n_{12} = 13 \cdot 8 \;(\mathrm{mod}\; 127) = 104 \\
p = 4: & n_{4\cdot 4\,(\mathrm{mod}\,14)} = n_0 \cdot 2^{4\cdot 1} \;(\mathrm{mod}\; 2^7 - 1) \Rightarrow n_2 = 13 \cdot 16 \;(\mathrm{mod}\; 127) = 81 \\
p = 5: & n_{5\cdot 4\,(\mathrm{mod}\,14)} = n_0 \cdot 2^{5\cdot 1} \;(\mathrm{mod}\; 2^7 - 1) \Rightarrow n_6 = 13 \cdot 32 \;(\mathrm{mod}\; 127) = 35 \\
p = 6: & n_{6\cdot 4\,(\mathrm{mod}\,14)} = n_0 \cdot 2^{6\cdot 1} \;(\mathrm{mod}\; 2^7 - 1) \Rightarrow n_{10} = 13 \cdot 64 \;(\mathrm{mod}\; 127) = 70
\end{array}$$

and, as usual, for $p = \frac{T}{lcm(\tau,T)} = 7$ we obtain the initial value $n_0 = 13$ as expected.

$$p = 7: \quad n_{7\cdot 4\,(\mathrm{mod}\,14)} = n_0 \cdot 2^{7\cdot 1} \;(\mathrm{mod}\; 2^7 - 1) \Rightarrow n_0 = 13 \cdot 128 \;(\mathrm{mod}\; 127) = 13$$

To sum up, all the information about a Bernoulli attractor can be retrieved from only one element of each subgroup of the orbit, like 31 and 49, and their basins of attraction (in this example the basins of attraction are formed by 15 bit strings altogether, as depicted in Fig. 7.10). Then, using only these 17 elements we are able to draw the whole basin-tree diagram, which is composed of 119 bit strings.

Note that the Bernoulli orbit shown in Fig. 7.8 can be interpreted as an attractor with $\sigma = 4$ and $\tau = 2$. It is easy to verify that this fact does not change the results given by Eq. 7.13, confirming that such formula is independent from the actual values of the parameters.

7.4 Scale-free property of additive rules

All non-trivial additive rules, presented in Sec. 6.4.4, are wither complex or hyper Bernoulli-shift rules; hence, their Bernoulli parameters σ and τ - and consequently, the periods of their orbits - depend crucially on L.
In principle, given L and the corresponding maximum period T_L of the orbits of a non-trivial additive rule, it is not possible to extract any information about the maximum period $T_{L'}$ of the orbit $L' \neq L$; in other words, the ratio $\frac{T_L}{L}$ is unrelated to $\frac{T'_L}{L'}$. Nevertheless, in [120] it was *empirically* noticed that for rule 90, 105, and 150 there exist certain values of L and L' for which

$$\frac{\log(T_{L'}) - \log(T_L)}{\log(L') - \log(L)} \Leftrightarrow \frac{T_{L'}}{L'} = \frac{T_L}{L}$$

This means that such rules exhibit a *scale-free* property as $L \to \infty$, and hence there are some $L' \neq L$ for which it is possible to know $T_{L'}$ by using only the information on T_L; the presence of a scale-free property also for rule 60 is confirmed in [137]. The scale-free property can be easily discovered at Figs. 7.11, 7.12, 7.12, and 7.12 taken from [138], and noticing that most points lie on diagonal lines with slope 1 (here we draw only a few of them).

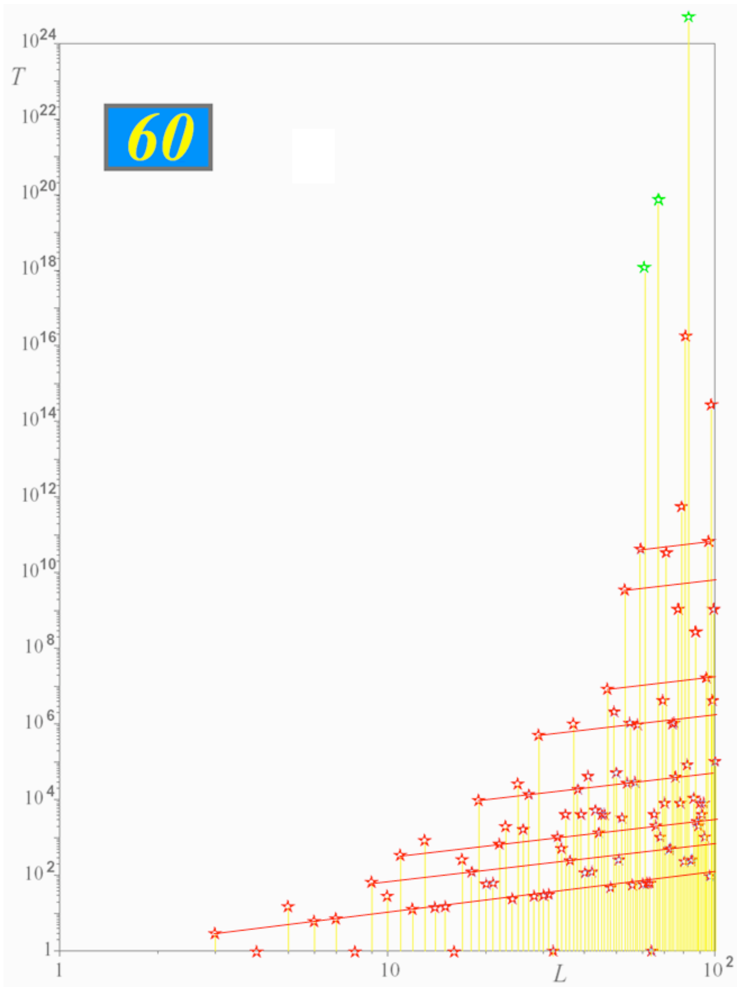


Figure 7.11: Attractors for $L \leq 100$ for rule 60: diagonal lines with slope 1 are indicative of the scale-free property. Green stars denotes points for which the attractor has a period exceeding the simulation time.

In the following we characterize quantitatively this phenomenon, finding out the values of L and $L\prime$ for which the scale-free property holds. We start by

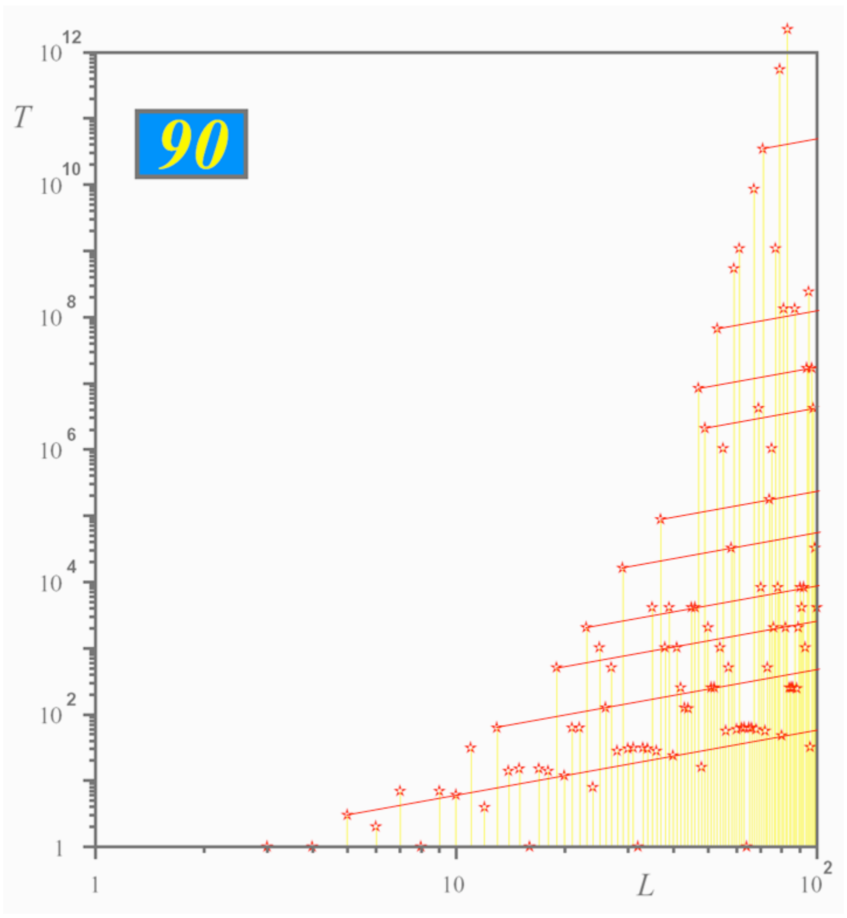


Figure 7.12: Attractors for $L \leq 100$ for rule 90: diagonal lines with slope 1 are indicative of the scale-free property.

giving the following definition:

Definition 7.4 (Scale-free order) *Let L be the length of the bit strings and let T^*_L be the maximum period of the orbits (attractors of Isles of Eden) of a non-trivial additive tule, then its scale-free order is defined as ξ_L*

$$\xi_L = \frac{T^*_L}{L} = \frac{2^{s(L)} - 1}{L}$$

Therefore, we need to find all L and L', with $L \neq L'$, for which $\xi_L = \xi_{L'}$. In the following we will make extensively use of the notation $o(L)$ and $s(L)$, indicating the multiplicative order and suborder, respectively, of the natural number L. A thorough explanation of these concepts is given in Appendix, along with the tables with their values for $L < 100$. Moreover, from now on, we will refer exclusively to rules 90 and 150 for which, as proved in [138] and reported in the Appendix, $T^*_L = \frac{2^{s(L)}-1}{L}$; however, the conclusions for rules 60 and 105 are very similar and can be achieved through a similar procedure. In the Appendinx are also tabulated the amximum period of the orbits for rule 60, 90, 105, and 150, which will be used in the examples of this section. The following theorem represents an important advance towards our goal.

Theorem 7.2 (Scale-free order) *Let L be the length of the bit strings and let ξ_L be its scale-free order, of $o(L) = s(L)$, then L can be uniquely be expressed as*

$$L = \frac{2^{o(\xi_L)}-1}{\xi_L} \sum_{i=0}^{n} s^{i \cdot o(\xi_L)} \tag{7.15}$$

where

$$n = \frac{o(L)}{o(\xi_L)} - 1$$

This theorem is very powerful because given any L for which $o(L) = s(L)$ and its scale-free order ξ_L, we can easily find analytically *all* the L' for which $\xi_L = \xi_{L'}$ by varing the parameter n.

Example 7.1 *For $L = 21$, the values of the multiplicative order and suborder are $o(21) = s(21) = 6$, and the maximum period of the orbit is $T^*_L = 63$; therefore, $\xi_L = \frac{T^*_L}{L} = 3$ and $o(\xi_L) = 2$. From Eq. 7.15,*

$$n = \frac{o(L)}{o(\xi)} - 1 = \frac{6}{2} - 1 = 2$$

and

$$n: 2 \;\rightarrow L = \frac{2^2-1}{3}(1 + 2^{1\cdot 2} + 2^{2\cdot 2}) = 21$$

as expected.
Considering the following values for the parameter n - i.e. $n = 3, 4, 5$, etc. - we can find an infinite number of $L \neq L'$ such that $\xi_L = \xi_{L'}$. For example,

$$\begin{array}{lll} n:3 & \rightarrow & L' = \frac{2^2-1}{3}(1 + 2^{1\cdot 2} + 2^{2\cdot 2}) = 21 \\ n:5 & \rightarrow & L' = \frac{2^2-1}{3}(1 + 2^{1\cdot 2} + 2^{2\cdot 2} + 2^{3\cdot 2}) = 85 \\ n:7 & \rightarrow & L' = \frac{2^2-1}{3}(1 + 2^{1\cdot 2} + 2^{2\cdot 2} + 2^{3\cdot 2} + 2^{4\cdot 2}) = 341 \\ & \cdots & \end{array}$$

Example 7.2 *For $L = 15$, the values of the multiplicative order and suborder are $o(15) = s(15) = 4$, and the maximum period of the orbit is $T_L^* = 15$; therefore, $\xi_L = \frac{T_L^*}{L} = 1$ and $o(\xi_L) = 1$. From Eq. 7.15,*

$$n = \frac{o(L)}{o(\xi)} - 1 = \frac{4}{1} - 1 = 3$$

and

$$n : 3 \ \rightarrow L = \frac{2^1 - 1}{1}(1 + 2^{1\cdot 1} + 2^{1\cdot 2} + 2^{1\cdot 3}) = 15$$

as expected.
In this case, we consider all values $n > 1$ - i.e. n=1, 2, 4, etc. - to be sure of finding all $L \neq L'$ such that $\xi_L = \xi_{L'} = 1$. For example,

$$\begin{array}{lll} n:1 & \rightarrow & L' = \frac{2^1-1}{1}(1+2^{1\cdot 1}) = 3 \\ n:2 & \rightarrow & L' = \frac{2^1-1}{1}(1+2^{1\cdot 1}+2^{2\cdot 1}) = 7 \\ n:4 & \rightarrow & L' = \frac{2^1-1}{1}(1+2^{1\cdot 1}+2^{1\cdot 2}+2^{3\cdot 1}+2^{4\cdot 1}) = 31 \\ & \dots & \end{array}$$

Example 7.3 *For $L = 73$, the values of the multiplicative order and suborder are $o(73) = s(73) = 9$, and the maximum period of the orbit is $T_L^* = 511$; therefore, $\xi_L = \frac{T_L^*}{L} = 7$ and $o(\xi_L) = 3$. From Eq. 7.15,*

$$n = \frac{o(L)}{o(\xi)} - 1 = \frac{9}{3} - 1 = 2$$

and

$$n : 2 \ \rightarrow L = \frac{2^7 - 1}{7}(1 + 2^{1\cdot 3}) = 73$$

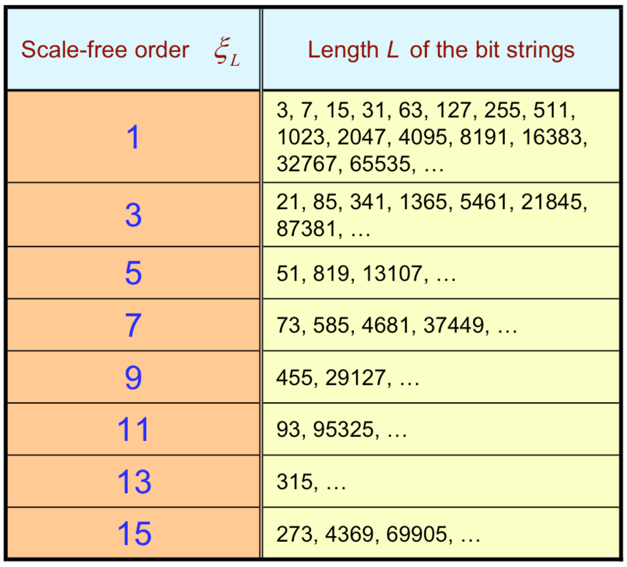
as expected.
If we 'go backward' using $n = 1$, a already done in the previous example, we apparently find a contradiction because

$$n : 1 \ \rightarrow \ L' = \frac{2^7-1}{7}(1 + 2^{1\cdot 3}) = 9$$

but the table of the orbit periods (see appendix) indicates that $T_9^* = 31$ and then $\xi_9 = 3.44 \neq \xi_{73}$. However, this happens because $o(9) = 6$ and $s(9) = 3$, and since $o(9) \neq s(9)$, Theorem 7.2 cannot be applied to $L = 9$.

The values of L corresponding to $\xi_L \leq 15$ are presented in Table **??**.
The results of this section can be summarized in the following theorem:

Table 7.1: Length L of the bit strings clssified according to their scale-free order.



Scale-free order ξ_L	Length L of the bit strings
1	3, 7, 15, 31, 63, 127, 255, 511, 1023, 2047, 4095, 8191, 16383, 32767, 65535, …
3	21, 85, 341, 1365, 5461, 21845, 87381, …
5	51, 819, 13107, …
7	73, 585, 4681, 37449, …
9	455, 29127, …
11	93, 95325, …
13	315, …
15	273, 4369, 69905, …

Theorem 7.3 (Scale-free property of additive rules) *Let L be the length of the bit strings and let T_L^* be the maximum period of the orbit of a non-trivial additive rule, then for any $L' \in \mathbb{N}$ the scale-free property*

$$\frac{T_{L'}^*}{L'} = \frac{T_L^*}{L}$$

holds if, and only if,

1. *both L and L' satisfy the conditions of Theorem 7.2, if L' is odd;*

2. *$L' = 2n \cdot L$, with $n \in \mathbb{N}$, if L' is even and $L, L' \neq 2^i$.*

Figure 7.15 gives a graphical interpretation of this theorem: red lines correspond to values of $\xi_L \in \mathbb{N}$, which will include all odd L obtained from Theorem 7.2 (as stated in the first point of Theorem 7.3) plus their even multiples (as stated in the second point of Theorem 7.3); blue lines correspond to values of $\xi_L \notin$, and then they include only one odd L (because L does not satisfy the conditions of Theorem 7.2) plus its even multiples (as stated in the second point of Theorem 7.3).

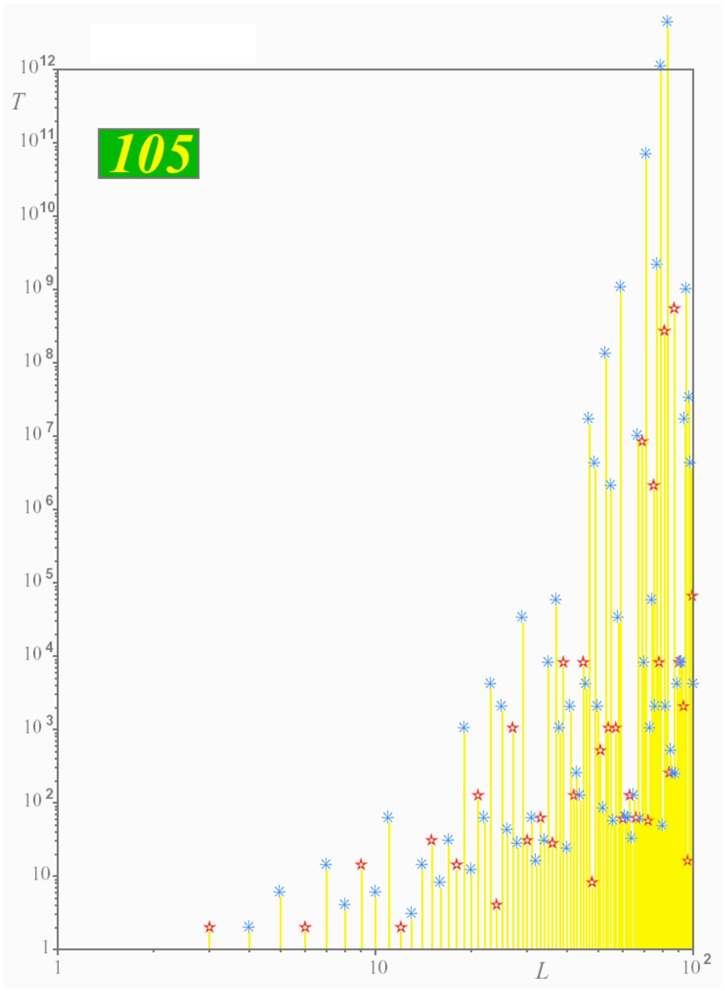


Figure 7.13: Attractors (red stars) and Isles of Eden (blue stars) for $L \leq 100$ for rule 105: diagonal lines with slope 1 are indicative of the scale-free property.

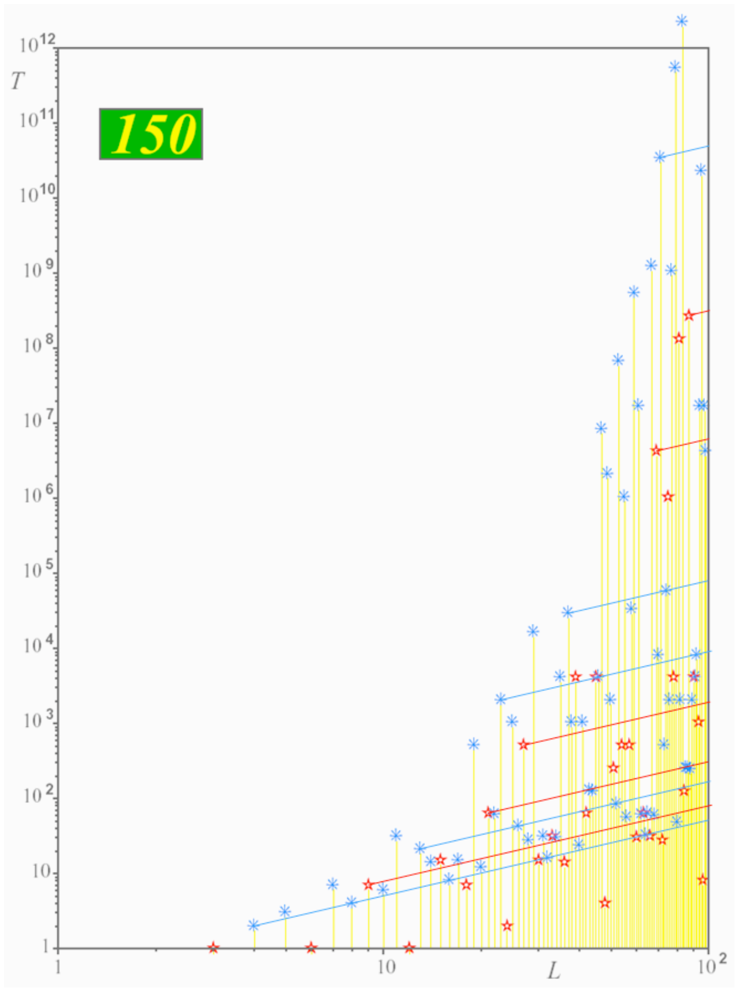


Figure 7.14: Attractors (red stars) and Isles of Eden (blue stars) for $L \leq 100$ for rule 150: diagonal lines with slope 1 are indicative of the scale-free property.

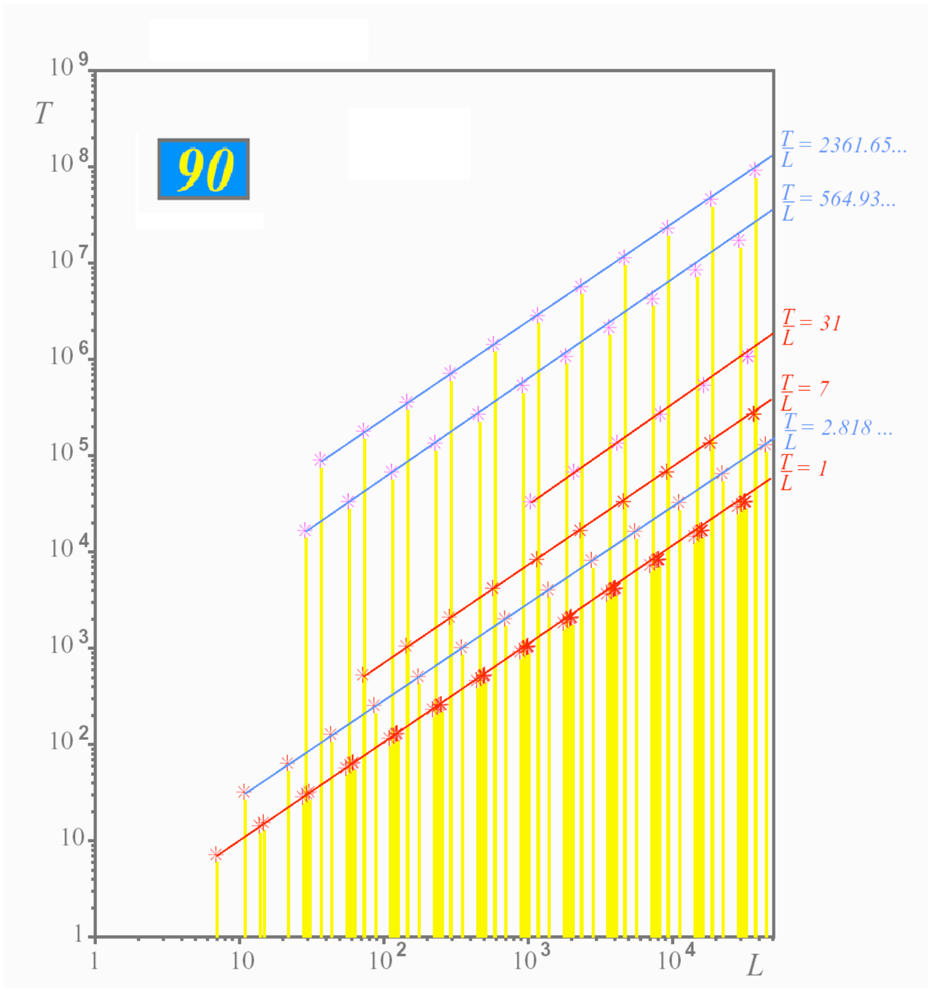


Figure 7.15: Scale-free property: when $\xi_L = \frac{T}{L} \in \mathbb{N}$ (in red) the line includes attractors for infinite values of L, L odd, plus their even multiples; when $\xi_L = \frac{T}{L} \notin \mathbb{N}$ (in blue) the line includes attractors for only one value of L, L odd, plus its even multiples

Chapter 8

Conclusions and future work

Our contribution to the theory of cellular paradigms can be divided into two main parts, one regarding algorithms for Cellular Wave Computers, and the other concerning the dynamics of Cellular Automata.

As for the first point, we started by proving that the Cellular Neural Network - Universal Machine is indeed universal. Although this fundamental result has been known for more than a decade, what presented here goes beyond the mere proof of universality, defining also an algorithmic structure common to all the CNN-UM programs.

These theoretical results has also remarkable practical consequences. For example, the search space defined by all the CNN-UM programs is now bounded and well-defined, and then it can be efficiently explored through machine learning techniques. In particular, we put forward Genetic Programming, which has has been successfully employed in similar scenarios. We proved that our the GP-based system meets the requirements of closure and sufficiency when dealing with CNN-UM programs, and we also analyzed in detail a number of other features related to this problem, like metrics to compare images and the appropriate choice of the genetic operators. Finally, we tested the effectivity of our approach on several experiments, showing how our system was capable of finding new efficient solutions in a variety of cases, including real-life situations.

Several conclusions can be drawn on this part of the work. First of all, our results about the form of CNN-UM programs are the first contribution in this field, as for our knowledge. Other authors explored the possibility of using genetic techniques to create automatically programs for the Cellular Neural Network - Universal Machine; however, the search space including all the possible CNN-UM programs is too huge to be explored even by fine artificial intelligence techniques, unless one does not define a standard form for all CNN-UM programs, as we did.

The second part of this dissertation was devoted to the analysis of Cellular Automata from a nonlinear dynamics perspective. An exhaustive description of this topic would be impossible here, then we focused on a few aspects which have seen our direct participation. We started illustrating Chua's classification of CA rules, based on the properties of their attractors, and then we explained fundamental techniques such as the time-τ characteristic map, the time-1 return plot, and the Lamerey (cobweb) diagram. Thanks to these tools, it has been possible to uncover fundamental features of CA, like quasi-ergodicity, the presence of fractal patterns, and the properties of the so-called Bernoulli orbits. Recently, this topic has become of interest of the scientific community, and it is possible to expect that more result will follow soon.

As often happens, many results have been found, but many more ideas and conjectures have remained to explore. Here, we express some of them to encourage further investigations into these topics.

The fact that the universality of the CNN-UM was proved without making use of the *Game of Life* is certainly important, however it would be interesting to explore how such proof can be applied directly on this paradigm, and not through an analogy with another universal machine. This may give further information about the form of the programs as well as which instructions are "fundamental" and which are not. A further idea would be studying in detail how CNN templates can be decomposed into "fundamental" ones, and defining a minimal set of instructions that implement all the others. This development may be then extended to devices, analyzing whether the approach "few instructions, complex programs" gives advantages with respect to "complex instructions, simple programs" in terms of execution time. The discussion would be somehow similar to the well-known dilemma of RISC and CISC architectures.

Also the system based on GP to create automatically CNN-UM programs can be improved, adding more functionalities and testing it on new examples. For instance, the inclusion of if-then-else statements in the programs is still not completely defined, and also the way in which memory is used could be changed, making the system more efficient and effective. We explored alternative techniques, like linear GP [139], without finding relevant advances. It would be interesting not to limit to Genetic Programming, but also trying completely different methods that may give better results.

Finally, the nonlinear perspective of CA seems to be very promising, and we are currently working on it. One of our short term objectives is proposing a way of describing rules alternative to Chua's Boolean cubes, and it may lead to important conclusions about the CA rule dynamics. We are also investigating whether the 88 classes in which rules are grouped can be reduced by using transformations weaker that the *Viergrouppe ones*, or limiting the analysis to particular cases.

Appendix A

Euler totient function, multiplicative order and suborder

Some of the results given in chapter 7 require the knowledge of three concepts of number theory, which we introduce here.

Definition A.1 (Euler totient function) *Give a positive integer n, the Euler totient function $\Phi(n)$ is the number of positive integers less than n that are coprime to n.*

Definition A.2 (Multiplicative order function) *Given a positive integer n, the multiplicative order of 2 (mod n) is the minimum positive integer $o(n)$ for which*

$$2^{o(n)} = 1 \ \textit{(mod n)}$$

Definition A.3 (Multiplicative suborder function) *Given a positive integer n, the multiplicative suborder of 2 (mod n) is the minimum positive integer $s(n)$ for which*

$$2^{s(n)} = \pm 1 \ \textit{(mod n)}$$

Remark A.1 *It is possible to prove that*

$$\Phi(n) \leq n - 1$$

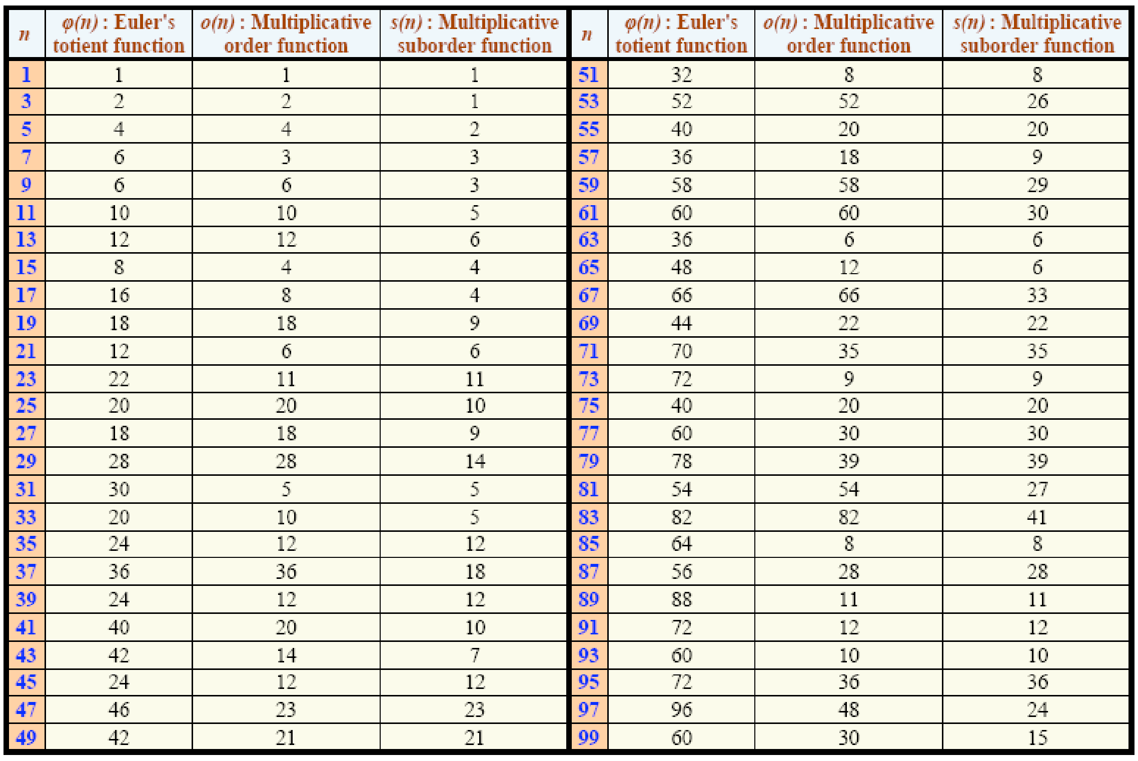
and

$$s(n) \leq \frac{n-1}{2}$$

and

$$s(n) | o(n) | \Phi(n)$$

Table A.1: Values of $\Phi(n)$, $o(n)$, and $s(n)$ for n odd and $n < 100$.



n	*φ(n)* : Euler's totient function	*o(n)* : Multiplicative order function	*s(n)* : Multiplicative suborder function	*n*	*φ(n)* : Euler's totient function	*o(n)* : Multiplicative order function	*s(n)* : Multiplicative suborder function
1	1	1	1	**51**	32	8	8
3	2	2	1	**53**	52	52	26
5	4	4	2	**55**	40	20	20
7	6	3	3	**57**	36	18	9
9	6	6	3	**59**	58	58	29
11	10	10	5	**61**	60	60	30
13	12	12	6	**63**	36	6	6
15	8	4	4	**65**	48	12	6
17	16	8	4	**67**	66	66	33
19	18	18	9	**69**	44	22	22
21	12	6	6	**71**	70	35	35
23	22	11	11	**73**	72	9	9
25	20	20	10	**75**	40	20	20
27	18	18	9	**77**	60	30	30
29	28	28	14	**79**	78	39	39
31	30	5	5	**81**	54	54	27
33	20	10	5	**83**	82	82	41
35	24	12	12	**85**	64	8	8
37	36	36	18	**87**	56	28	28
39	24	12	12	**89**	88	11	11
41	40	20	10	**91**	72	12	12
43	42	14	7	**93**	60	10	10
45	24	12	12	**95**	72	36	36
47	46	23	23	**97**	96	48	24
49	42	21	21	**99**	60	30	15

The values of $\Phi(n)$, $o(n)$, and $s(n)$ for n odd and $n < 100$ are listed in Table A.1.
It is possible to prove the following theorems [138] that

Theorem A.1 (Maximum period of the orbits of rules 90 and 150) *Let L be the length of the bit strings and let T_L be the maximum period of the orbit (attractor or Isle of Eden) of rules 90 and 150, then*

1. *If L is odd, then T_L divides the quantity $T^* = 2^{s(L)} - 1$, where $s(L)$ is the multiplicative suborder of 2 (mod n);*
2. *If $L = 2^n$, then $\mathbf{x} = (00\ldots0)$ is a global attractor;*
3. *If L is even but not of the form $L = 2^n$, then $T_L = 2 \cdot T_{\frac{L}{2}}$*

Theorem A.2 (Maximum period of the orbits of rule 60) *Let L be the length of the bit strings and let T_L be the maximum period of the orbit (attractor or Isle of Eden) of rule 60, then*

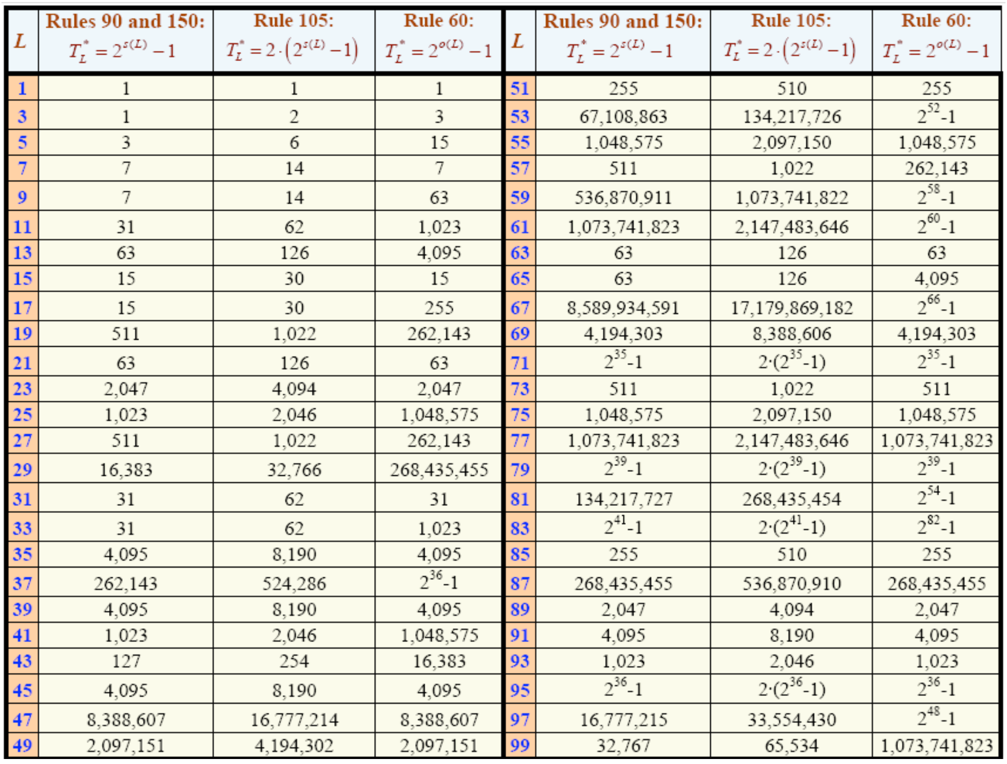
1. *If L is odd, then T_L divides the quantity $T^* = 2^{o(L)} - 1$, where $o(L)$ is the multiplicative order of 2 (mod n);*
2. *If $L = 2^n$, then $\mathbf{x} = (00\ldots0)$ is a global attractor;*
3. *If L is even but not of the form $L = 2^n$, then $T_L = 2 \cdot T_{\frac{L}{2}}$*

Theorem A.3 (Maximum period of the orbits of rule 105) *Let L be the length of the bit strings and let T_L be the maximum period of the orbit (attractor or Isle of Eden) of rule 105, then*

1. *If L is odd, then T_L divides the quantity $T^* = 2 \cdot (2^{s(L)} - 1)$, where $s(L)$ is the multiplicative suborder of 2 (mod n);*
2. *If L is even, then T_L divides the quantity $T^*_L = 2 \cdot (2^{s(\frac{L}{2})} - 1)$*

It is important to notice that the parameter T^*_L does not give necessarly the maximum period T_L of the orbits, because in general $T_L | T^*_L$; however, we found experimentally that $T_L = T^*_L$ for most L. The values of T^*_L for rules 60, 90, 105, and 150 are given in Table A.2

Table A.2: Values of T_L^* for rules 60, 90, 105, and 150.



L	Rules 90 and 150: $T_L^* = 2^{s(L)} - 1$	Rule 105: $T_L^* = 2 \cdot \left(2^{s(L)} - 1\right)$	Rule 60: $T_L^* = 2^{o(L)} - 1$	L	Rules 90 and 150: $T_L^* = 2^{s(L)} - 1$	Rule 105: $T_L^* = 2 \cdot \left(2^{s(L)} - 1\right)$	Rule 60: $T_L^* = 2^{o(L)} - 1$
1	1	1	1	51	255	510	255
3	1	2	3	53	67,108,863	134,217,726	2^{52}-1
5	3	6	15	55	1,048,575	2,097,150	1,048,575
7	7	14	7	57	511	1,022	262,143
9	7	14	63	59	536,870,911	1,073,741,822	2^{58}-1
11	31	62	1,023	61	1,073,741,823	2,147,483,646	2^{60}-1
13	63	126	4,095	63	63	126	63
15	15	30	15	65	63	126	4,095
17	15	30	255	67	8,589,934,591	17,179,869,182	2^{66}-1
19	511	1,022	262,143	69	4,194,303	8,388,606	4,194,303
21	63	126	63	71	2^{35}-1	2·(2^{35}-1)	2^{35}-1
23	2,047	4,094	2,047	73	511	1,022	511
25	1,023	2,046	1,048,575	75	1,048,575	2,097,150	1,048,575
27	511	1,022	262,143	77	1,073,741,823	2,147,483,646	1,073,741,823
29	16,383	32,766	268,435,455	79	2^{39}-1	2·(2^{39}-1)	2^{39}-1
31	31	62	31	81	134,217,727	268,435,454	2^{54}-1
33	31	62	1,023	83	2^{41}-1	2·(2^{41}-1)	2^{82}-1
35	4,095	8,190	4,095	85	255	510	255
37	262,143	524,286	2^{36}-1	87	268,435,455	536,870,910	268,435,455
39	4,095	8,190	4,095	89	2,047	4,094	2,047
41	1,023	2,046	1,048,575	91	4,095	8,190	4,095
43	127	254	16,383	93	1,023	2,046	1,023
45	4,095	8,190	4,095	95	2^{36}-1	2·(2^{36}-1)	2^{36}-1
47	8,388,607	16,777,214	8,388,607	97	16,777,215	33,554,430	2^{48}-1
49	2,097,151	4,194,302	2,097,151	99	32,767	65,534	1,073,741,823

Bibliography

[1] S. Boggan and D. M. Pressel, "GPUs: An emerging platform for general-purpose computation," U.S. Army Research Lab., Tech. Rep. ARL-SR-154, Aug. 2007.

[2] A. Kis, F. Kovacs, and P. Szolgay, "3D tactile sensor array processed by CNN-UM: A fast method for detecting and identifying slippage and twisting motion," *International Journal of Circuit Theory and Applications*, vol. 34, pp. 517–531, 2006.

[3] W. Porod, F. Werblin, L. Chua, T. Roska, A. R. Vasquez, B. Roska, P. Fay, G. H. Bernstein, Y. F. Huang, and A. I. Csurgay, "Bio-inspired nano-sensor-enhanced CNN visual computer," *Annals of the New York Academy of Sciences*, vol. 1013, pp. 92–109, 2004.

[4] D. Bálya, B. Roska, T. Roska, and F. S. Werblin, "A CNN framework for modeling parallel processing in a mammalian retina: Research articles," *International Journal of Circuit Theory and Applications*, vol. 30, no. 2-3, pp. 363–393, 2002.

[5] S. Espejo, Á. Rodríguez-Vázquez, R. Domínguez-Castro, J. Huertas, and E. Sánchez-Sinencio, "Smart-pixel cellular neural networks in analog current-mode CMOS technology," *IEEE J. Solid-State Circuits*, vol. 29, no. 8, pp. 895–905, Aug. 1994.

[6] International technology roadmap for semiconductors. (2008) 2007 ITRS report, emerging research materials. [Online]. Available: http://www.itrs.net/Links/2007ITRS/2007_Chapters/2007_ERM.pdf

[7] "Overview of the IBM blue gene/p project," *IBM Journal of Research and development*, vol. 52, no. 1/2, 2008.

[8] J. A. Kahle et al., "Introduction to the cell multiprocessor," *IBM Journal of Research and development*, vol. 49, no. 4/5, 2005.

[9] S. R. Vangal et al., “An 80-tile sub-100-W teraFLOPS processor in 65-nm CMOS,” *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 29–41, Jan. 2008.

[10] L. Chua and L. Yang, “Cellular neural networks: theory,” *IEEE Trans. Circuits Syst.*, vol. 35, pp. 1257–1272, Oct. 1988.

[11] ——, “Cellular neural networks: applications,” *IEEE Trans. Circuits Syst.*, vol. 35, pp. 1273–1290, Oct. 1988.

[12] M. Balsi, “Focal-plane optical flow computation by foveated cnns,” in *Proc. of Fifth IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA-98*, 1998, pp. 14–17.

[13] M. Itoh and L. O. Chua, “Star cellular neural networks for associative and dynamic memories,” *International Journal of Bifurcation and Chaos*, vol. 14, no. 5, pp. 1725–1772, 2004.

[14] A. G. Radvanyi, “On the rectangular grid representation of general CNN networks,” *International Journal of Circuit Theory and Applications*, vol. 30, no. 2-3, pp. 181–193, 2002.

[15] L. O. Chua and T. Roska, “The CNN paradigm,” *IEEE Trans. Circuits Syst. I*, vol. 40, no. 3, pp. 147–156, Mar. 1993.

[16] H. Harrer and J. A. Nossek, “Discrete-time cellular neural networks,” *International Journal of Circuit Theory and Applications*, vol. 20, pp. 453–467, 1992.

[17] S. Espejo, R. Carmona, R. Domínguez-Castro, and Á. Rodríguez-Vázquez, “A VLSI-oriented continuous-time CNN model,” *International Journal of Circuit Theory and Applications*, vol. 24, no. 3, pp. 341–356, 1996.

[18] L. Kék, K. Karacs, and T. Roska. (2007) Cellular wave computing library, version 2.1 (templates, algorithms and programs). [Online]. Available: http://cnn-technology.itk.ppke.hu/Library_v2.1b.pdf

[19] S. Majorana and L. Chua, “A unified framework for multilayer high order CNN,” *International Journal of Circuit Theory and Applications*, vol. 26, no. 6, pp. 567–592, 1998.

[20] T. Roska and L. O. Chua, “The CNN universal machine: an analogic array computer,” *IEEE Trans. Circuits Syst. II*, vol. 40, pp. 163–173, Mar. 1993.

[21] K. Karacs, “Text recognition, event detection and some theoretical aspects of cellular wave computer architectures,” Ph.D. dissertation, Pázmány Péter Catholic University, Budapest, 2007.

[22] L. Chua and T. Roska, “The cellular neural network (CNN) and the CNN universal machine: concept, architecture and operation modes,” in *Towards the visual microprocessor*, T. Roska and Á. Rodríguez-Vázquez, Eds. Chichester, UK: Wiley, 2001.

[23] T. Roska, “Cellular wave computers for nano-tera-scale technology — beyond Boolean, spatial-temporal logic in million processor devices,” *Electronics letters*, vol. 43, no. 8, 2007.

[24] J. von Neumann, *The Computer and the Brain.* New Haven: Yale University Press, 1966.

[25] A. Turing, “The chemical basis of morphogenesis,” *Philosophical Transactions of the Royal Society of London*, vol. 237, pp. 37–72, 1952.

[26] S. Ulam, “Processes and transformations,” in *Proc. of International Congress of Mathematics*, vol. 2, 1952, pp. 264–265.

[27] J. von Neumann, *The theory of self-reproducing automata.* Urbana, IL: University of Illinois Press, Urbana, 1966.

[28] S. Wolfram, *A new kind of science.* Wolfram media, Inc, 2002.

[29] L. Chua, *Nonlinear Dynamics Perspective of Wolfram’ New Kind of Science.* World scientific, 2007.

[30] L. O. Chua, V. I. Sbitnev, and S. Yook, “A nonlinear dynamics perspective of Wolframs new kind of science. part III: Predicting the unpredictable,” *International Journal of Bifurcation and Chaos*, vol. 14, no. 11, pp. 3689–3820, 2004.

[31] P. Földesy, A. Zarandy, C. Rekeczky, and T. Roska, “High performance processor array for image processing,” in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS 2007)*, New Orleans, CA, May 27–30, 2007, pp. 1177–1180.

[32] “Q-eye data sheet,” AnaFocus SA, Seville, Spain.

[33] (2006) Analogic and neural computing laboratory of the hungarian academy of sciences. [Online]. Available: http://lab.analogic.sztaki.hu/

[34] (2006) The microelectronic institute of Seville. [Online]. Available: http://www.imse.cnm.es/

[35] (2008) Anafocus. [Online]. Available: http://www.anafocus.com/

[36] (2008) Eutecus. [Online]. Available: http://www.eutecus.com/

[37] Cellular Sensory Wave Computing Laboratory. (2008) Software for simulating cellular neural networks. [Online]. Available: http://www.analogic.sztaki.hu/index.php?a=downloads&c=2

[38] L. Yang, L. Chua, and K. Krieg, "VLSI implementation of cellular neural networks," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS'90)*, vol. 3, May 1–3, 1990, pp. 2425–2427.

[39] H. Harrer, J. Nossek, and R. Stelzl, "An analog implementation of discrete-time cellular neural networks," *International Journal of Circuit Theory and Applications*, vol. 3, no. 3, pp. 466–476, May 1992.

[40] J. Cruz and L. Chua, "A CNN chip for connected component detection," *IEEE Trans. Circuits Syst.*, vol. 38, pp. 812–817, July 1991.

[41] A. Gruss, L. Carley, and T. Kanade, "Integrated sensors and range-finding analog signal processor," *IEEE J. Solid-State Circuits*, vol. 26, pp. 184–191, Mar. 1991.

[42] R. Domínguez-Castro *et al.*, "A 0.8μm CMOS programmable mixed-signal focal-plane array processor with on-chip binary imaging and instructions storage," *IEEE J. Solid-State Circuits*, vol. 32, pp. 1013–1026, July 1997.

[43] P. Földesy, A. Zarandy, P. Szolgay, and T. Szirányi, "A real-life application case studies using CMOS 0.8μm CNN universal chip: Analogic algorithm for motion detection and texture segmentation," in *Proc. fourth IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA'96)*, Seville, Spain, June 24–26, 1996, pp. 363–368.

[44] G. L. Cembrano, S. Espejo-Meana, R. Domínguez-Castro, and Á. Rodríguez-Vázquez, "ACE4k: An analog I/O 64x64 visual microprocessor chip with 7-bit analog accuracy," *International Journal of Circuit Theory and Applications*, vol. 30, pp. 89–116, 2002.

[45] Á. Rodríguez-Vázquez *et al.*, "ACE16k: The third generation of mixed-signal SIMD-CNN ACE chips toward VSoCs," *IEEE Trans. Circuits Syst. I*, vol. 51, no. 5, pp. 851–863, May 2004.

[46] Ángel Rodríguez-Vázquez et al., "The Eye-RIS CMOS vision system," in *Analog Circuit Design*, M. S. Herman Casier and A. H. M. V. Roermund, Eds. Springer Netherlands, 2008.

[47] R. Carmona, F. Jiménez-Carrido, R. Domínguez-Castro, S. Espejo-Meana, and Á. Rodríguez-Vázquez, "CMOS realization of a 2-layer CNN universal machine chip," in *Proc. of 2002 7th IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA'02)*, Frankfurt, Germany, July 22–24, 2002, pp. 444–451.

[48] A. Adamatzky *et al.*, "Reaction-diffusion navigation robot control: from chemical to VLSI analogic processors," *IEEE Trans. Circuits Syst. I*, vol. 51, no. 5, pp. 926–938, May 2004.

[49] A. Zarandy *et al.*, "Various implementations of topographic, sensory, cellular wave computers," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS'05)*, Kobe, 2005, pp. 5802–5805.

[50] Z. Nagy and P. Szolgay, "Configurable multilayer CNN-UM emulator on FPGA," *IEEE Trans. Circuits Syst. I*, vol. 50, no. 6, pp. 774–778, June 2003.

[51] P. Keresztes *et al.*, "An emulated digital CNN implementation," *The Journal of VLSI Signal Processing*, vol. 23, no. 2-3, pp. 291–303, Nov. 1999.

[52] T. Hidvegi, P. Keresztes, and P. Szolgay, "An accelerated digital CNN-UM (CASTLE) architecture by using the pipe-line tecnique," in *Proc. of 2002 7th IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA'02)*, Frankfurt, Germany, July 22–24, 2002, pp. 355–362.

[53] Z. Nagy, Z. Voroshazi, and P. Szolgay, "An emulated digital retina model implementation on FPGA," in *Proc. of 2005 9th IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA2005)*, May 28–30, 2005, pp. 278–281.

[54] C. Baldanza *et al.*, "A cellular neural network for peak finding in high energy physics," in *Proc. of 2000 6th IEEE International Workshop on*

Cellular Neural Networks and their Applications (CNNA 2000), May 23–25, 2000, pp. 443–448.

[55] P. Kozma, P. Sonkoly, and P. Szolgay, "Elastic wave propagation modeling on emulated digital CNN-UM architecture," in *Proc. of 2005 9th IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA 2005)*, June 8–10, 2005, pp. 906–912.

[56] F. J. Toledo, J. J. Martínez, F. J. Garrigós, and J. M. Ferrández, "Image processing with cnn in a FPGA-based augmented reality system for visually impaired people," in *Proc. of 8th International Work-Conference on Artificial Neural Networks (IWANN 2005)*, Barcelona, Spain, May 28–30, 2005, pp. 126–129.

[57] G. E. Pazienza et al., "Optimized cellular neural network universal machine emulation on FPGA," in *Proc. 2007 European Conference on Circuit Theory and Design (ECCTD'07)*, Seville, Spain, 2007.

[58] G. E. Pazienza, X. Ponce-García, M. Balsi, and X. Vilasís-Cardona, "Robot vision with cellular neural networks: a practical implementation of new algorithms," *International Journal of Circuit Theory and Applications*, vol. 35, no. 4, pp. 449–462, 2007.

[59] K. Oh and K. Jung, "GPU implementation of neural networks," *Pattern Recognition*, vol. 37, no. 6, pp. 1311–1314, 2004.

[60] W. B. Langdon and W. Banzhaf, "A SIMD interpreter for genetic programming on GPU graphics cards," in *EuroGP*, ser. LNCS. Naples: Springer, 26-28 Mar. 2008.

[61] C. Rekeczky. (2006) MATCNN: analogic CNN simuation toolbox for MATLAB. [Online]. Available: http://lab.analogic.sztaki.hu/Candy/matcnn.html

[62] B. Soos, A. Rak, J. Veres, and G. Cserey, "GPU powered cnn simulator SIMCNN with graphical ow based programmability," in *Proc. 11th International Workshop on Cellular Neural Networks and their Applications (CNNA'08)*, Santiago de Compostela, Spain, July 14-16 2008, pp. 163–168.

[63] A. Fernandez, R. S. Martin, E. Farguell, and G. E. Pazienza, "Cellular neural networks simulation on a parallel graphics processing unit," in *Proc. 11th International Workshop on Cellular Neural Networks and*

their Applications (CNNA'08), Santiago de Compostela, Spain, July 14-16 2008, pp. 208–212.

[64] NVidia. (2008) CUDA programming documentation. [Online]. Available: http://www.nvidia.com/object/cuda_develop.html

[65] J. Koza, *Genetic programming - on the programming of computers by means of natural selection.* Cambridge, MA: MIT-Press, 1992.

[66] R. Poli, W. B. Langdon, and N. F. McPhee, *A field guide to genetic programming.* Lulu.com, 2008, (With contributions by J. R. Koza). [Online]. Available: http://www.gp-field-guide.org.uk

[67] A. Turing, "On computable numbers, with an application to the entscheidungsproblem," *Proceedings of the London Mathematical Society*, vol. 2, no. 42, pp. 230–265, 1936.

[68] J. E. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages and Computation.* Reading, Mass.: Addison-Wesley, 1979.

[69] R. Feynman, *Feynman lectures on computation.* Reading, Mass.: Addison-Wesley, 1996.

[70] L. O. Chua, T. Roska, and P. Venetianer, "The CNN is as universal as the Turing machine," *IEEE Trans. Circuits Syst. I*, vol. 40, no. 4, pp. 289–291, Apr. 1993.

[71] L. Chua and B. Shi, "Exploiting cellular automata in the design of cellular neural networks for binary image processing," UC Berkeley, Electron. Res. Lab., CA, Tech. Rep. UCB/ERL M89/130, Nov. 1989.

[72] R. Dogaru and L. Chua, "Universal CNN cells," *International Journal of Bifurcation and Chaos*, vol. 9, no. 1, pp. 1–48, 1999.

[73] G. E. Pazienza, E. Gómez-Ramírez, , and X. Vilasís-Cardona, "Polynomial cellular neural networks for implementing the game of life," in *Proc. International Conference on Artificial Neural Networks (ICANN'07)*, Porto, Portugal, Sept.9-13 2007.

[74] Z. Galias, "Designing cellular neural networks for the evaluation of local Boolean functions," *IEEE Trans. Circuits Syst. II*, vol. 40, pp. 219–223, Mar. 1993.

[75] C. Rekeczky, T. Roska, and A. Ushida, "CNN-cased difference-controlled adaptative non-linear image filters," *International Journal of Circuit Theory and Applications*, vol. 26, pp. 375–423, 1998.

[76] E. Berlekamp, J. H. Conway, and R. K. Guy, *Winning ways for your mathematical plays.* New York: Academic Press, 1982.

[77] P. Rendell. (2006) A Turing machine in Conway's game life. Available: www.cs.ualberta.ca/ ˜ bulitko/f02/papers/tmwords.pdf.

[78] A. Teller, "Turing completeness in the language of genetic programming with indexed memory," in *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, vol. 1. Orlando, Florida, USA: IEEE Press, 27-29 June 1994, pp. 136–141.

[79] J. McCarthy, "Recursive functions of symbolic expressions and their computation by machine, part i," *Communications of the ACM*, vol. 3, no. 4, pp. 184–195, 1960.

[80] S. Handley, "On the use of a directed acyclic graph to represent a population of computer programs," in *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, vol. 1, Orlando, Florida, USA, 27-29 June 1994, pp. 154–159.

[81] R. Diestel, *Graph Theory*, 3rd ed. Heidelberg, Germany: Springer-Verlag, 2005.

[82] A. Zarandy, F. Werblin, T. Roska, and L. Chua, "Novel types of analogic CNN algorithms for recognizing banknotes," in *Proc. third IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA'94)*, Rome, Italy, Dec. 18–21 1994, pp. 273–278.

[83] P. Venetianer, F. Werblin, L. O. Chua, and T.Roska, "Analogic CNN algorithms for some image compression and restoration tasks," *IEEE Trans. Circuits Syst. I*, vol. 42, no. 5, 1995.

[84] A. Schultz, I. Szatmári, C. Rekeczky, T. Roska, and L. Chua, "Bubble-debris classification via binary morphology and autowave metric on CNN," in *Proc. International symposium on Nonlinear theory and its applications*, Hawaii, US, 1997.

[85] V. Preciado, D. Guinea, J. Vicente, M. García-Alegre, and A. Ribeiro, "Automatic CNN multi-template tree-generation," in *Proc. 6th IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA'00)*, Catania, Italy, Dec. 18–21 2000, pp. 381–385.

[86] ——, "Genetic programming of a CNN multi-template tree for automatic generation of analogic algorithms," in *Proc. 6th IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA'00)*, Catania, Italy, Dec. 18–21 2000, pp. 327–332.

[87] V. Preciado, M. Preciado, and M. Jaramillo, *Genetic Programming for Automatic Generation of Image Processing Algorithms on the CNN Neuroprocessing Architecture.* Springer, 2004, vol. 3040, pp. 374–383.

[88] P. López, D. López-Vilarino, and D. Cabello, "Design of multilayer discrete time cellular neural networks for image processing tasks based on genetic algorithms," in *Proc. of 2000 IEEE International Symposium on Circuits and Systems (ISCAS 2000)*, vol. 4, Geneva, Switzerland, May 28–31 2000, pp. 133–136.

[89] P. López, M. Balsi, D. Lopez-Vilarino, and D. Cabello, "Design and training of multilayer discrete time neural networks for antipersonnel mine detection using genetic algorithms," in *Proc. of 2000 6th IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA 2000)*, Catania, Italy, May 23–25 2000, pp. 363–368.

[90] G. E. Pazienza, E. Gómez-Ramírez, and X. Vilasís-Cardona, "Genetic programming for the CNN-UM," in *Proc. 10th International Workshop on Cellular Neural Networks and their Applications (CNNA'06)*, Istanbul, Turkey, Aug. 28-30 2006.

[91] G. E. Pazienza, K. Karacs, and X. Vilasís-Cardona, "An automatic tool to design CNN-UM programs," in *Proc. 2007 European Conference on Circuit Theory and Design (ECCTD'07)*, Seville, Spain, 2007.

[92] G. E. Pazienza, R. Poli, and X. Vilasís-Cardona, "An alternative proof of universality of the CNN-UM and its practical consequences," in *Proc. 11th International Workshop on Cellular Neural Networks and their Applications (CNNA'08)*, Santiago de Compostela, Spain, July 14-16 2008.

[93] P. Angeline and J. B. Pollack, "Competitive environments evolve better solutions for complex tasks," in *Proc. 5th International Conference on Genetic Algorithms (GA-93)*, 1993, pp. 264–270.

[94] W. B. Langdon, "Better trained ants," in *Late Breaking Papers at EuroGP'98: the First European Workshop on Genetic Programming*, R. Poli, W. B. Langdon, M. Schoenauer, T. Fogarty, and W. Banzhaf, Eds., Paris, France, 14-15 Apr. 1998, pp. 11–13.

[95] C. Gathercole, "An investigation of supervised learning in genetic programming," Ph.D. dissertation, University of Edinburgh, 1998.

[96] M. Fuchs, "Large populations are not always the best choice in genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference*, vol. 2, Orlando, Florida, USA, 13-17 July 1999, pp. 1033–1038.

[97] C. Gathercole and P. Ross, "Small populations over many generations can beat large populations over few generations in genetic programming," in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, Stanford University, CA, USA, 13-16 July 1997, pp. 111–118.

[98] A. E. Magurran, *Ecological diversity and its measurement.* Princeton, NJ: Princeton University Press, 1988.

[99] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, "Genetic programming: An introductory tutorial and a survey of techniques and applications," University of Essex, UK, Tech. Rep. CES-475, Oct. 2007.

[100] D. Huttenlocher, G. Klanderman, and W. Rucklidge, "Comparing images using the Hausdorff distance," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 15, no. 9, pp. 850–863, Sept. 1993.

[101] V. Biktashev, V. Krinsky, and H. Haken, "A wave approach to pattern recognition (with application to optical character recognition)," *International Journal of Bifurcation and Chaos*, vol. 4, no. 1, pp. 193–207, 1994.

[102] I. Szatmari, A. Schultz, C. Rekeczky, T. Kozek, T. Roska, and L. Chua, "Morphology and autowave metric on CNN applied to bubble-debris classification," *IEEE Trans. Neural Networks*, vol. 11, no. 6, pp. 1385–1393, Nov. 2000.

[103] I. Szatmari, "Nonlinear wave metric for object comparison on CNN arcitecture," Ph.D. dissertation, Hungarian Academy of Sciences, Budapest, 2001.

[104] G. Borgefors, "Distance transforms in arbitrary dimensions," *Computer Vision, Graphics and Image Processing*, vol. 27, pp. 321–345, 1984.

[105] S. Silva. (2006) GPLAB - a genetic programming toolbox for MATLAB. [Online]. Available: http://gplab.sourceforge.net/

[106] Y. Lin and J. Hsieh, "Robust template decomposition with restricted weigts for cellular neural networks implementing and arbitrary Boolean function," *International Journal of Bifurcation and Chaos*, vol. 17, no. 9, pp. 3151–3169, 2007.

[107] M. Laiho, A. Paasio, J. Flak, and K. Halonen, "Template design for cellular nonlinear networks with one-bit weights," *IEEE Trans. Circuits Syst. I*, to be published.

[108] T. Szirany and M. Csapodi, "Texture classification and segmentation by cellular neural network using genetic learning," *Computer vision and Image Understanding*, vol. 71, no. 3, pp. 255–270, 1998.

[109] K. Karacs and T. Roska, "Route number recognition of public transport vehicles via the bionic eyeglass," in *Proc. 10th International Workshop on Cellular Neural Networks and their Applications (CNNA'06)*, Istanbul, Turkey, Aug. 28-30 2006.

[110] F. Liu and N. Goldenfeld, "Generic features of late-stage crystal growth," *Phys. Rev. A*, vol. 42, no. 895903, 1990.

[111] J. Moreira and A. Deutsch, "Cellular automaton models of tumor development: A critical review," *Advances in Complex System*, vol. 5, no. 2-3, pp. 247–269, 2002.

[112] R. Hegselmann and A. Flache, "Understanding complex social dynamics: A plea for cellular automata based modelling," *Journal of Articial Societies and Social Simulation*, vol. 1, no. 3, 1998.

[113] F. Schweitzer, L. Behera, and H. M*ü*hlenbei, "Evolution of cooperation in a spatial prisoners dilemma," *Advances in Complex System*, vol. 5, no. 2-3, pp. 269–301, 2002.

[114] S. Wolfram, *Cellular Automata and complexity (Collected papers).* Addison-Wesley, 1994.

[115] L. O. Chua, S. Yook, and R. Dogaru, "A nonlinear dynamics perspective of Wolfram's new kind of science. Part I: Threshold of complexity," *International Journal of Bifurcation and Chaos*, vol. 12, no. 12, pp. 2655–2766, 2002.

[116] L. O. Chua, V. I. Sbitnev, and S. Yook, "A nonlinear dynamics perspective of Wolfram's new kind of science. Part II: Universal neuron," *International Journal of Bifurcation and Chaos*, vol. 13, no. 9, pp. 2377–2491, 2003.

[117] I. Niven, *Irrational numbers.* The Mathematical association of America, 1967.

[118] L. P. Shilnikov, A. L. Shilnikov, D. Turaev, and L. Chua, *Methods of Qualitative Theory in Nonlin- ear Dynamics: Part I.* Singapore: World Scientic, 1998.

[119] K. T. Alligood, T. Sauer, and J. Yorke, *Chaos: An Introduction to Dynamical Systems.* Springer-Verlag NY, 1996.

[120] L. O. Chua, J. Guan, V. I. Sbitnev, and J. Shin, "A nonlinear dynamics perspective of Wolfram's new kind of science. Part VII: Isles of Eden," *International Journal of Bifurcation and Chaos*, vol. 17, no. 9, pp. 2839–3125, 2007.

[121] L. O. Chua, V. I. Sbitnev, and S. Yook, "A nonlinear dynamics perspective of Wolfram's new kind of science. Part IV: From Bernoulli shift to 1/f spectrum," *International Journal of Bifurcation and Chaos*, vol. 15, no. 4, pp. 1045–1183, 2005.

[122] E. F. Moore, "Machine models of self-reproduction," *Proc. Symp. Appl. Math.*, vol. 17, pp. 17–33, 1962.

[123] G. M. B. Oliveira, P. P. B. de Oliveira, and N. Omar, "Definition and application of a five-parameter characterization of one-dimensional cellular automata rule space," *Artif. Life*, vol. 7, no. 3, pp. 277–301, 2001.

[124] G. Cattaneo, E. Formenti, L. Margara, and G. Mauri, "On the dynamical behavior of chaotic cellular automata," *Theor. Comput. Sci.*, vol. 217, no. 1, pp. 31–51, 1999.

[125] C. Domain and H. Gutowitz, "The topological skeleton of cellular automaton dynamics," *Phys. D*, vol. 103, no. 1-4, pp. 155–168, 1997.

[126] G. Braga, G. Cattaneo, P. Flocchini, and C. Quaranta Vogliotti, "Pattern growth in elementary cellular automata," *Theor. Comput. Sci.*, vol. 145, no. 1-2, pp. 1–26, 1995.

[127] J. A. de Sales, M. L. Martins, and J. G. Moreira, "One-dimensional cellular automata characterization by the roughness exponent," *Physica A*, vol. 245, no. 3, pp. 461–471, 1997.

[128] V. C. Barbosa, F. M. N. Miranda, and M. C. M. Agostini, "Cell-centric heuristics for the classification of cellular automata," *Parallel Comput.*, vol. 32, no. 1, pp. 44–66, 2006.

[129] J. C. Dubacq, B. Durand, and E. Formenti, "Kolmogorov complexity and cellular automata classification," *Theor. Comput. Sci.*, vol. 259, no. 1-2, pp. 271–285, 2001.

[130] K. Culik, L. P. Hurd, and S. Yu, "Computation theoretic aspects of cellular automata," *Phys. D*, vol. 45, no. 1-3, pp. 357–378, 1990.

[131] D. Eppstein. (2008) Gliders and Wolfram's classification. [Online]. Available: http://www.ics.uci.edu/ eppstein/ca/wolfram.html

[132] K. Culik and S. Yu, "Undecidability of CA classification schemes," *Complex Syst.*, vol. 2, no. 2, pp. 177–190, 1988.

[133] M. Courbage, D. Mercier, and S. Yasmineh, "Traveling waves and chaotic properties in cellular automata," *Chaos*, vol. 9, no. 4, pp. 893–901, 1999.

[134] F. Ohi, "Chaotic properties of elementary CA rule 168," in *Automata 2008: EPSRC Workshop Cellular Automata Theory and Applications*, Bristol, UK, June12–14 2008.

[135] M. Shereshevsky, "Ergodic properties of certain surjective cellular automata," in *Monatshefte für Mathematik.* Springer Wien, 1992.

[136] M. Shirvani and T. Rogers, "On ergodic one-dimensional cellular automata," *Communications in Mathematical Physics*, vol. 136, pp. 599–605, 1991.

[137] L. O. Chua, K. Karacs, V. I. Sbitnev, J. Guan, and J. Shin, "A nonlinear dynamics perspective of Wolfram's new kind of science. Part VIII: More isles of Eden," *International Journal of Bifurcation and Chaos*, vol. 17, no. 9, pp. 2839–3125, 2007.

[138] L. O. Chua, G. E. Pazienza, L. Ozro, V. I. Sbitnev, and J. Shin, "A nonlinear dynamics perspective of Wolfram's new kind of science. Part IX: Quasi-ergodicity," *International Journal of Bifurcation and Chaos*, 2008.

[139] M. F. Brameier and W. Banzhaf, *Linear Genetic Programming.* Springer, 2007.

Universitat Ramon Llull

Aquesta Tesi Doctoral ha estat defensada el dia ____ d ___________________ de ____

al Centre ____

de la Universitat Ramon Llull

davant el Tribunal format pels Doctors sotasignants, havent obtingut la qualificació:

President/a

____

Vocal

____

Vocal

____

Vocal

____

Secretari/ària

____

Doctorand/a

C. Claravall, 1-3
08022 Barcelona
Tel. 936 022 200
Fax 936 022 249
E-mail: urlsc@sec.url.es
www.url.es