Deducing Local Rules for Solving Global Tasks with Random Boolean Networks

Bertrand Mesot [a] Christof Teuscher [b,1]

[a] *IDIAP Research Institute, Rue de Simplon 4, CH–1920, Martigny, Switzerland*

[b] *Los Alamos National Laboratory, Advanced Computing Laboratory, CCS-1, MS-B287, Los Alamos, NM 87545, USA*

Abstract

It has been shown that uniform as well as non-uniform cellular automata (CA) can be evolved to perform certain computational tasks. Random Boolean networks are a generalization of two-state cellular automata, where the interconnection topology and the cell's rules are specified at random.

Here we present a novel analytical approach to find the local rules of random Boolean networks (RBNs) to solve the global density classification and the synchronization task from any initial configuration. We quantitatively and qualitatively compare our results with previously published work on cellular automata and show that randomly interconnected automata are computationally more efficient in solving these two global tasks. Our approach also provides convergence and quality estimates and allows the networks to be randomly rewired during operation, without affecting the global performance. Finally, we show that RBNs outperform small-world topologies on the density classification task and that they perform equally well on the synchronization task.

Our novel approach and the results may have applications in designing robust complex networks and locally interacting distributed computing systems for solving global tasks.

Key words: random Boolean network, cellular automata, density classification task, synchronization task, small-world topologies
PACS:

Email addresses: `bertrand.mesot@idiap.ch` (Bertrand Mesot), `christof@teuscher.ch` (Christof Teuscher).

URLs: `http://www.idiap.ch/~bmesot` (Bertrand Mesot), `http://www.teuscher.ch/christof` (Christof Teuscher).

[1] The work in this paper was completed while affiliated with the University of California, San Diego (UCSD), and supported by grant PBEL2–104420 from the

1 Introduction

Cellular automata (CA) [51, 55] were originally conceived by Ulam and von Neumann [53] in the 1940s to provide a formal framework for investigating the behavior of complex, extended systems. CAs are dynamical systems in which space and time are discrete. A cellular automaton usually consists of a D-dimensional regular lattice of N lattice sites, commonly called *cells*. Each cell can be in one of a finite number of S possible states and further consists of a transition function F (also called *rule*), which maps the neighboring states to the set of cell states. CAs are called *uniform* if all cells contain the same rule, otherwise they are *non-uniform*. Each cell takes as input the states of the cells within some finite local neighborhood. By convention, a cell is considered to be a member of its own neighborhood. In the standard one-dimensional CA model, for example, each cell is connected to the r (r stands for radius) immediate local neighbors on either side as to itself, thus each cell is connected to $2r + 1$ neighbors.

Random Boolean networks (RBNs), on the other hand, form a more general class of discrete dynamical systems, in which two-state cellular automata are a special subclass. Random networks and RBNs were investigated in the past forty years by many a researcher (see for example [2, 4, 7, 36]), but became popular mainly due to the contributions of Stuart Kauffman [24–26] and others.

In its simplest form, a RBN is composed of N nodes (sometimes also called *elements* or *cells*) where each node can be in one of two possible states (0 or 1) and receives inputs from K randomly chosen other nodes (self-connections are allowed). Note that K can refer to the *exact* or to the *average* number of connections between the nodes. In the geneticist's term, for example, K measures the richness of epistatic interactions among the components of a system [26].

The deterministic behavior of each node is specified by one out of the 2^{2^K} Boolean functions, specifying its next value for each of the 2^K combinations with K inputs. Like CAs, RBNs can be *non-uniform* (i.e., each node can potentially have a different rule) or *uniform*, although they are non-uniform in the majority of the cases. The network's nodes are usually updated synchronously, although multiple asynchronous updating schemes exist (see [20] for an overview).

Synchronous random Boolean networks as introduced by Kauffman are commonly called NK networks or models. Figure 1 shows a possible NK random Boolean network representation ($N = 8, K = 3$). The model is very similar

Swiss National Science Foundation.

to the well-studied class of models which arises in statistical physics, called *spin-glasses* [18]. Spin-glasses are disordered magnetic materials in which the orientation of nearby magnetic dipoles may be either parallel or antiparallel in space.

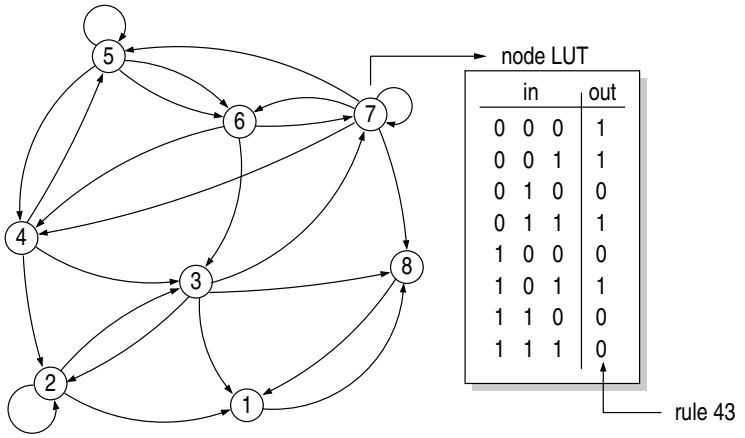

Fig. 1. Illustration of a random Boolean network with $N = 8$ nodes and $K = 3$ inputs per node (self-connections are allowed). The node rules are commonly represented by *lookup-tables* (LUTs), which associate a 1-bit output (the node's future state) to each possible K-bit input configuration. The table's out-column is commonly called the *rule* of the node.

More formally speaking, let $\mathbf{s}^t$ be the state-vector of the network's nodes at time t. The neighborhood $\Pi(x_i)$ of a node x_i is defined as

$$\Pi(x_i) \subset P_K(\{x_1, \ldots, x_N\}),$$

where $P_K(X)$ denotes the set of all subsets of size K from X (self-connections are allowed). The neighborhood size is given by $K = |\Pi(x_i)|$. The most common ways to choose the neighborhood are the following:

- *random neighborhood*: choose the K neighbors randomly from the set $\{x_1, \ldots, x_N\} \setminus \{x_i\}$ (without self-connections) or from the set $\{x_1, \ldots, x_N\}$ (self-connections allowed)
- *adjacent neighborhood*: K neighbors are randomly chosen in the "immediate" neighborhood.

$A(N, K)$ is sometimes used to denote a NK model with adjacent neighborhood, $N(N, K)$ for the random neighborhood. We shall not go into more details of the adjacent neighborhood here, as we will only consider purely random networks with a uniform probability to connect two nodes together in this article.

Figure 1 shows the interaction graph $G(V, E)$ where $V = \{x_1, \ldots, x_N\}$ corresponds to the set of nodes, and $\{x_i, x_j\} \in E$ if and only if x_i is connected to x_j. The node rules are commonly represented by *lookup-tables* (LUTs), which

associate a 1-bit output (the node's future state) to each possible K-bit input configuration. The table's out-column is commonly called the *rule* of the node.

Whereas the homogeneous connectivity and the usually uniform rules of a CA restrict the automaton's parameter space, RBNs have a vastly greater space, which adds an additional challenge for both evolutionary and analytical methods, either to analyze the networks' behavior or to find networks able to perform a given task.

The main contribution of this paper consists in a novel analytical method which allows to determine the local rules of a random Boolean network for two global and well-known problems for cellular automata, (1) the synchronization and (2) the density classification task. Whereas previous work was purely quantitative and focused on the co-evolution of cellular automata to find suitable rules and architectures [45], our qualitative approach goes a step further and also provides convergence and quality estimates. We systematically compare the results with previous work on standard and non-standard cellular automata architectures, including small-world networks, and show that randomly interconnected automata perform better than a standard cellular automata with a regular interconnection architecture. Our method also allows the networks to be randomly rewired during operation without affecting the global performance. The approach and results may have applications in designing robust complex networks and locally interacting distributed computing systems for solving global tasks.

In Section 2 we describe the two global tasks used in this paper, the density classification task and the synchronization task. The main challenge in solving such global tasks with CAs and RBNs consists in finding the node's local rules that will result in the desired global automata behavior. Note that no generally applicable method for this exists. In Section 3 we derive the recursive equation which gives the probability of a node to be in state 1 at the next time step. This equation is then used in Section 4 to determine the node's rules for both tasks. Section 5 examines the network's performance as a function of N and K whereas Section 5.3 compares the results with previously published performance results of cellular automata for the same tasks. In order to measure the rule's capacity to solve the tasks, we introduce an entropy-based performance measure in Section 5.4. The same measure also allows us to predict how quickly the two tasks can be solved. In Section 6 we briefly discuss non-uniform random Boolean networks and perform experiments with small-world network topologies. Our findings, their possible future applications, and future work are discussed in Section 7.

2 Task Definitions

Two commonly used applications for CAs are the *density* and the *synchronization* task. Both of these "global" tasks are mostly trivial to solve if one has a global view on the system (i.e., if one has access to the state of all nodes at the same time), but are non-trivial to solve for CA and RBNs, mainly because of the locality and the limited number of their interconnections, the simplicity of the basic cells (i.e., the basic components), and because there is no generally valid way to determine the cells's rules for such a massive parallel system. Unlike in the standard approach to parallel computation, in which a given problem is split into independent sub-problems, CAs and RBNs have to solve problems by the bottom-up approach, where the global and usually complex behavior arises from nonlinear, spatially extended, and local interactions [30]. This difficulty has naturally also limited CAs applications.

In the following, the density classification and the synchronization task shall be briefly described.

2.1 The Density Classification Task

The density classification task for CAs must decide whether or not the initial configuration of the automaton contains more than 50% of 1s. In this context, the term "configuration" refers to an assignment of the states 0 or 1 to each cell of the CA (i.e., there are 2^N possible initial configurations). The desired behavior of the automaton is to have all of its cells set to 1 if the initial density of 1s exceeded 1/2, and all 0 otherwise. The density task for CAs can straightforwardly be applied to RBNs. The special case of having an equal number of 1s and 0s in the network is commonly avoided by using an odd number of nodes.

The density task was studied among others by Mitchell et al. [29–31], Das et al. [15,16], Sipper [38–41], Sipper et al. [43,45,46], Capcarrère [14], Capcarrère et al. [10–12] and Tomassini et al. [52] in various forms, including non-uniform CAs, asynchronous CAs, and non-standard architectures. It should also be mentioned that no uniform two-state CA exists, which perfectly solves the density classification task for all initial configurations [27].

Let n_1^t be the number of nodes being in state 1 at time t. The density classi-

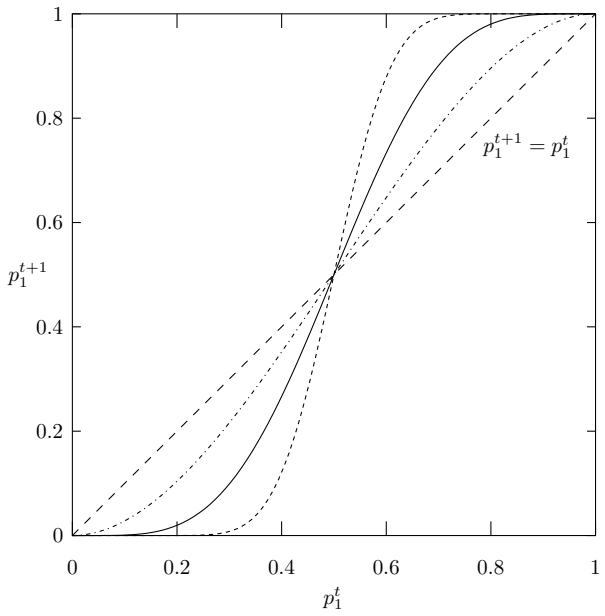

Fig. 2. Three possible maps which satisfy Equation 1. The dash-dotted curve represents the one that will take the highest number of iterations before settling down in a fixed point.

fication task can then be formally described by the following equation:

$$\exists t_{\text{lim}} \quad \text{s.t.} \quad t \geqslant t_{\text{lim}} \quad \Rightarrow \quad n_1^t = \begin{cases} 0 \ \text{ if } n_1^0 < N/2 \\ N \ \text{ if } n_1^0 > N/2 \ . \\ \dfrac{N}{2} \ \text{otherwise} \end{cases} \tag{1}$$

Let us define $p_1^t = n_1^t/N$ as the fraction of nodes that are in state 1 at time t. We will sometimes use p_1^t instead of n_1^t when the networks are sufficiently large (i.e., $N \to +\infty$), since p_1^t may then be interpreted as the probability of the node to be in state 1 at time t. Moreover, since we only consider a synchronous updating scheme, p_1^{t+1} will solely depend on p_1^t. The network's behavior can thus simply be described by a map of the form $p_1^{t+1} = Q(p_1^t)$, where $Q : \mathbb{R} \longmapsto \mathbb{R}$. Each of the three solutions of the density classification task will then be represented by a fixed point in this map. Figure 2 shows three possible functions Q that satisfy Equation 1. Note that the fixed point in $p_1^* = 1/2$ is unstable, which means that a small perturbation of the network's state, such as for example one state change among the N nodes, will drive the system towards the wrong fixed point, i.e., towards the wrong solution. This unstable fixed point can fortunately be avoided by using an odd number of nodes.

The one-dimensional *synchronization task* (also called *firefly task*) for synchronous CA was introduced by Das et al. [15] and studied among others by Hordijk [21] and Sipper [42]. In this task, the two-state one-dimensional or two-dimensional CA, given any initial configuration, must reach a final configuration within M time steps, that oscillates between all 0s and all 1s on successive time steps. The whole automaton is then globally synchronized.

As with the density classification task, synchronization is a non-trivial global computation task for a small-radius CA where all cells must coordinate their behavior with all the other cells while having only a very local view of their neighborhood. The two-state machine of each cell obviously also prevents any counting, which would make the task much less difficult. Furthermore, in the non-uniform case, where each cell can have a different rule, there is an immediate solution consisting of a unique "master" rule that alternates between 0 and 1 without looking at his neighbors, and all other cells being its "slave", i.e., alternating according to their right (or upper, in the two-dimensional case) neighbor state only. However, Sipper [42] used non-uniform CAs to find perfect synchronizing CAs by means of evolution only. It appeared that this very particular solution was never found by evolution and, in fact, the "master" or "blind" rule `10101010` (rule 170) in one dimension, was never part of the evolved solutions. This is simply due to the fact that this rule has to be unique for the solution to be perfect, which is contradictory to the natural tendency of the evolutionary algorithm used, as was demonstrated by Capcarrère [12].

The two-dimensional version of the task is identical to the one-dimensional version, except that the necessary number of time steps granted to synchronize is not anymore in the order of N, the number of cells in the automaton, but in the order of $n + m$, where n and m are the size of each side of the standard CA. Therefore, the speed of synchronization is much faster compared to the one-dimensional version.

Extending the two-dimensional synchronization task for CAs to RBNs is again straightforward and solutions found be means of co-evolutionary algorithms were presented by Teuscher and Capcarrère in [50]. The firefly synchronization task was also successfully implemented in actual evolutionary hardware, for both the one [44] and the two-dimensional [50] version.

Similarly to the density classification task, the synchronization task may also be formally described by a condition on the number of 1s in the network's

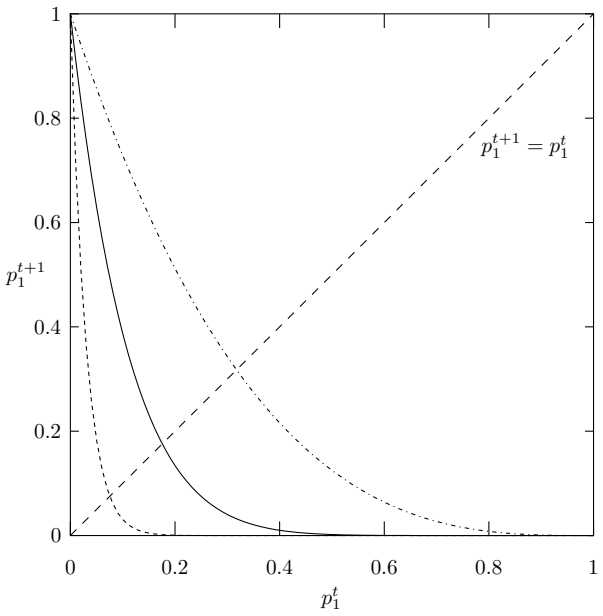

Fig. 3. Three possible maps which satisfy Equation 2. The dash-dotted curve requires the highest number of iterations before oscillating between all $p_1 = 0$ and $p_1 = 1$.

state:

$$\exists t_{\text{lim}} \quad \text{s.t.} \quad t \geqslant t_{\text{lim}} \quad \Rightarrow \quad n_1^t = \begin{cases} N \text{ if } n_1^{t-1} = 0 \\ 0 \text{ if } n_1^{t-1} = N \end{cases} \tag{2}$$

Figure 3 shows three possible maps which satisfy Equation 2. Contrary to the density classification task, fixed points must be avoided since oscillations are required.

3 From Global Configurations to Local Interactions

The challenge in solving the density classification and synchronization task with CAs and RBNs consists in finding the node's local rules that will result in a global automaton behavior which satisfies Equations 1 or 2 for all possible initial configurations. The difficulty comes from the fact that each node has only access to K randomly chosen bits of the entire RBN's state vector $\mathbf{s}^t$ at time t. It is therefore crucial to know how the global network state is seen by the local nodes in order to understand the network's dynamics.

Let W_1^t be a random variable representing the number of 1s in a node's neighborhood at time t and let N_1^t and $P_1^t = N_1^t/N$ be two random variables which represent the number of 1s and the percentage of 1s in the network's state $\mathbf{s}^t$ at

time t respectively. W_1^t obviously depends on P_1^t, thus, if we suppose that the nodes' neighbors are randomly chosen from a uniform distribution, then the probability of having d neighbors in state 1 at time t, knowing the percentage of 1s (i.e., p_1^t) in the global network state $\mathbf{s}^t$ of a RBN, is therefore given by:

$$P(W_1^t = d \,|\, P_1^t = p_1^t) = \binom{K}{d} (p_1^t)^d \, (1 - p_1^t)^{K-d}. \tag{3}$$

The distribution of W_1^t knowing P_1 thus follows a binomial law $\mathcal{B}(K, p_1^t)$ and the expected number of 1s in a node's neighborhood is therefore Kp_1^t, which implies that the average number of 1s in the input of a node is the same as the average number of 1s in the global network state. This property is actually the key that will allow us to reduce the complex behavior of N nodes to the stochastic behavior of a single node which behaves on the average like the entire network.

Let $\mathbf{s}^t$ be the state of a RBN at time t, composed of N nodes, each being randomly connected to K other nodes (self-connections are allowed). Furthermore, let us assume that N is sufficiently large to consider $p_1^t = n_1^t/N$ as the probability of a node to be in state 1 at time t. We will later see in our experiments that this approximation only affects performance. Knowing $\mathbf{s}^t$, the probability that the future state of node i is 1 is then given by:

$$P(S_i^{t+1} = 1 \,|\, \mathbf{S}^t = \mathbf{s}^t) = \sum_{j=0}^{2^K-1} P(S_i^{t+1} = 1 \,|\, \Phi_i^t = j) \, P(\Phi_i^t = j \,|\, \mathbf{S}^t = \mathbf{s}^t), \tag{4}$$

where S_i^{t+1} and Φ_i^t are two random variables that represent the state of node i at time $t+1$ and its input at time t respectively. This is a probabilistic description of the node's Boolean transfer function (i.e., its rule), where $P(\Phi_i^t \,|\, \mathbf{S}^t)$ and $P(S_i^{t+1} \,|\, \Phi_i^t)$ are the input and output distributions respectively. If $f_i(j)$ is the output of node i for an input value j, then the previous equation becomes:

$$P(S_i^{t+1} = 1 \,|\, \mathbf{S}^t = \mathbf{s}^t) = \sum_{j=0}^{2^K-1} f_i(j) \, P(\Phi_i^t = j \,|\, \mathbf{S}^t = \mathbf{s}^t), \tag{5}$$

since the probability of a node to be in state 1 at time $t+1$ is 1 if $f_i(j) = 1$, and 0 otherwise. By using the variables W_1^t and P_1^t as introduced above, this latter equation can further be reduced to:

$$P(S_i^{t+1} = 1 \,|\, P_1^t = p_1^t) = \sum_{j=0}^{K} c_j \, P(W_1^t = j \,|\, P_1^t = p_1^t), \tag{6}$$

where j is the number of 1s in the input of node i and c_j is the probability that

the future state (the output) of node i is 1, provided that the input contains j times a 1.

Putting Equation 3 into Equation 6 gives us:

$$P(S_i^{t+1} = 1 \,|\, P_1^t = p_1^t) = \sum_{j=0}^{K} c_j \binom{K}{j} \left(p_1^t\right)^j \left(1 - p_1^t\right)^{K-j}, \tag{7}$$

which holds for any node i. Since $P(S_i^{t+1} = 1 \,|\, P_1^t = p_1^t)$ is equivalent to p_1^{t+1}, the probability for a node to be in state 1 at time $t+1$, we can now express p_1^{t+1} as a function of p_1^t by means of the following recursive equation:

$$p_1^{t+1} = \sum_{j=0}^{K} a_j \left(p_1^t\right)^j \left(1 - p_1^t\right)^{K-j} \quad \text{with} \quad a_j = \binom{K}{j} c_j \in \left[0, \binom{K}{j}\right], \tag{8}$$

where the parameters a_j represent the number of the rule's inputs that contain j times a 1 and that produce a 1 as output. Note that, if we represent a node's rule by a lookup-table, then a_j corresponds to the number of entries that contain j times a 1 and that have a 1 as output. For example, the two-bit *XOR* rule (i.e., the output is 1 if and only if the two inputs are different) corresponds to $a_0 = 0$, $a_1 = 2$ and $a_2 = 0$ since the output is 1 if there is only one 1 in the input. On the other hand, the two-bit *NAND* rule corresponds to: $a_0 = 1$, $a_1 = 2$ and $a_2 = 0$.

Recently, Matache and Heidel [28] used a formula to describe the probability of finding a node in state 1 at time t to analyze deterministic chaos in random Boolean networks. The basic idea is similar to our approach, but their work, which extends the model studied by Andrecut and Ali [6], is limited to rule 126 only, whereas our approach is valid for any rules.

4 Finding the Node's Rules

4.1 Rules for the Density Classification Task

As seen in Section 2.1, solving the density task can be reduced to finding maps $p_1^{t+1} = Q_K(p_1^t)$ which satisfy the constraints that were informally represented in Figure 2. More formally speaking, these constraints may be defined by the following set of equations:

$$\text{(C1)} \qquad Q_K(0) = 0, \quad Q_K(1) = 1, \quad Q_K(1/2) = 1/2, \quad Q_K''(1/2) = 0$$

(C2) $\quad Q''_K(x) > 0 \quad \forall x \in [0, 1/2[\quad \text{and} \quad Q''_K(x) < 0 \quad \forall x \in]1/2, 1].$

However, according to Equation 8, we have:

$$Q_K(x) = \sum_{j=0}^{K} a_j \, x^j \, (1-x)^{K-j} \quad \text{with} \quad a_j \in \left[0, \binom{K}{j}\right]. \tag{9}$$

Applying constraints C1 and C2 to Equation 9 and solving it as a function of a_j provides us now with an elegant means to directly determine the node's rules for the density task. In the following sections, different possible solutions for small connectivity parameters K shall be presented. We will see that the simple *majority rule* presents a common solution for all networks with a connectivity $K > 2$.

4.1.1 $K = 2$: No Solution

For $K = 2$, Equation 9 becomes:

$$Q_2(x) = a_0(1-x)^2 + a_1 x(1-x) + a_2 x^2.$$

After applying constraints C1 we obtain the following set of values:

$$\begin{aligned} Q_2(0) = 0 \quad &\Rightarrow \quad a_0 = 0 \\ Q_2(1) = 1 \quad &\Rightarrow \quad a_2 = 1 \\ Q_2(1/2) = \frac{1}{2^2}(a_1 + 1) = \frac{1}{2} \quad &\Rightarrow \quad a_1 = 1 \end{aligned}$$

This implies $Q_2(x) = x$ and therefore $Q''_2(x) = 0$ for all x, which does not satisfy constraints C2. Hence, for $K = 2$ no set of rules is able to perfectly solve the density classification task for all initial configurations.

4.1.2 $K = 3$: A First Solution

For $K = 3$, Equation 9 becomes:

$$Q_3(x) = a_0(1-x)^3 + a_1 x(1-x)^2 + a_2 x^2(1-x) + a_3 x^3.$$

In order to satisfy constraints C1, we must impose $a_0 = 0$, $a_3 = 1$ and $a_2 = 3 - a_1$. $Q_3(x)$ and $Q''_3(x)$ then become:

$$Q_3(x) = 2(a_1 - 1)x^3 + 3(1 - a_1)x^2 + a_1 x$$
$$Q_3''(x) = 12(a_1 - 1)x + 6(1 - a_1).$$

After applying constraints C2, we obtain:

$$Q_3''(x) > 0 \quad \Rightarrow \quad 12(a_1 - 1)x > 6(a_1 - 1) \quad \Rightarrow \quad x < \frac{1}{2} \quad \text{iif} \quad a_1 = 0.$$

One can easily verify that $a_1 = 0$ is the sole solution. The corresponding polynomial is therefore:

$$Q_3(x) = 3x^2(1 - x) + x^3. \tag{10}$$

This solution simply corresponds to the well-known *majority* rule: its output is 1 if and only if there are more 1s than 0s in the inputs, and 0 otherwise. As one can verify, since $a_0 = a_1 = 0$, the only input with all bits set to 0 and the $\binom{K}{1} = 3$ inputs with one bit set to 1 result in a 0 as output. Conversely, for all inputs containing more than one bit set to 1, the output becomes 1, since $a_2 = 3$ and $a_3 = 1$.

4.1.3 $K = 4$: Many Rules

For $K = 4$ only one polynomial satisfies both sets of constraints. It corresponds to: $a_0 = 0$, $a_1 = 0$, $a_2 = 3$, $a_3 = 4$ and $a_4 = 1$. Since $a_2 = 3$ and according to Equation 9, of the $\binom{K}{2} = 6$ inputs with two bits set to 1, only half of them will set the output to 1. Since there are 20 possibilities of setting 3 out of 6 outputs to 1, there are 20 (out of $2^{16} = 65536$) equivalent rules which will successfully solve the density classification task. These rules are represented in Table 1. As one can see, all rules are in a certain sense *majority* rules, but applied to an even number of connections, reason why many possibilities exist.

4.1.4 $K = 5$: Many Polynomials

By the same procedure we find the values of a_j when $K = 5$:

$$a_0 = 0, \quad a_1 = 0, \quad a_2 \in [0, 5], \quad a_3 = 10 - a_2, \quad a_4 = 5 \quad \text{and} \quad a_5 = 1.$$

Since a_2 is the only free parameter, 6 polynomials satisfy conditions C1 and C2, each corresponding to a value of a_2. The *majority* rule is the only possible rule when $a_2 = 0$, whereas in the other cases, each polynomial represents many rules. For example, when $a_2 = 3$, $\binom{10}{3}\binom{10}{7} = 14,400$ equivalent rules exist.

Binary	Hex	Binary	Hex
1110100011101000	E8E8	1111100011100000	F8E0
1110101010101000	EAE8	1110111011000000	EEC0
1110110010101000	ECA8	1111101011000000	FAC0
1111100010101000	F8A8	1111111010000000	FE80
1110100011001000	E8C8	1111110011000000	FCC0
1110101001001000	EA48	1111110010100000	FCA0
1110110011001000	ECC8	1111110010001000	FC88
1111100011001000	F8C8	1111101010100000	FAA0
1110101011100000	EAE0	1111101010001000	FA88
1110110011100000	ECE0	1110111010100000	EEA0

Table 1
The 20 (out of $2^{16} = 65536$) rules for $K = 4$ random Boolean networks that successfully solve the density classification task for all initial configurations.

As one can see, representing the rules by means of a polynomial bears the advantage that rules with similar dynamics are grouped together by Equation 9. They may thus more easily be handled and analyzed on a meta-level.

4.2 Rules for the Synchronization Task

The procedure to find the rules for the synchronization task is similar to the density classification task. Again, we are looking for maps $p_1^{t+1} = Q_K(p_1^t)$ similar to those shown on Figure 3. More formally, the constraints we have to impose on $Q_K(x)$ are firstly

$$Q_K(0) = 1 \quad \text{and} \quad Q_K(1) = 0,$$

which will force the network to oscillate between all 0s and all 1s on consecutive time steps. And secondly, in order to make sure that the network's state converges towards $n_1^t = 0$ or $n_1^t = 1$, $Q''_K(x)$ needs to be monotone and $Q''_K(x) \neq 0$ for all $x \in [0, 1]$.

Two functions satisfying these constraints are for example:

$$Q_K(x) = (1 - x)^K \quad \text{and} \quad Q_K(x) = \sum_{i=0}^{K-1} x^i \, (1 - x)^{K-i}.$$

These functions are symmetrical and therefore perform equally well. We will call them γ_K-rules. Their behavior is very simple: the lookup-table associated with this rule outputs a 1 (respectively 0) if and only if all input bits are 0 (respectively 1). Examples are rule $\gamma_3 = 1$ and rule $\gamma_3 = 128$ for $K = 3$ networks. Both rules are basically fully equivalent and solve the synchronization task in the same way.

5 Experiments and Results

In this section we will examine the performance of RBNs as a function of the number of nodes N and the number of connections per node K. Since the number of initial configurations exponentially grows with N, testing them all becomes rapidly computationally intractable. Algorithm 1 illustrates the procedure we used for all experiments in this section. Since the nodes are randomly connected, we average the simulation results on a certain number of randomly generated networks. In addition, each initial configuration is generated according to a biased distribution, i.e., the number of 1s it contains is randomly chosen from a uniform distribution and the 1s are then assigned to randomly chosen nodes. This will force the network to be tested uniformly on all possible values of p_1^0 and not only around $p_1^0 = 1/2$, which is the case if each configuration is chosen according to an unbiased distribution, i.e., each node has a uniform probability to be in state 1 or 0. Finally, in order to compare

Algorithm 1 Performance Evaluation Algorithm

```
N_min = 9 (minimal number of nodes)
N_max = 201 (maximal number of nodes)
rep = 5000 (number of initial configurations)
nnet = 500 (number of networks)
for n ∈ [N_min, N_max] do
  for i ∈ [1, nnet] do
    Generate a random network composed of n nodes.
    for r ∈ [1, rep] do
      Generate an initial configuration.
      Test the configuration on the network.
    end for
    Average the results over rep.
  end for
  Average the results over nnet.
end for
```

RBNs with CAs, we shall only test networks with an odd number of nodes. This avoids the unstable fixed point in $p_1^0 = 1/2$, which will be misclassified most of the time anyway, since, as explained in Section 2.1, a small perturba-

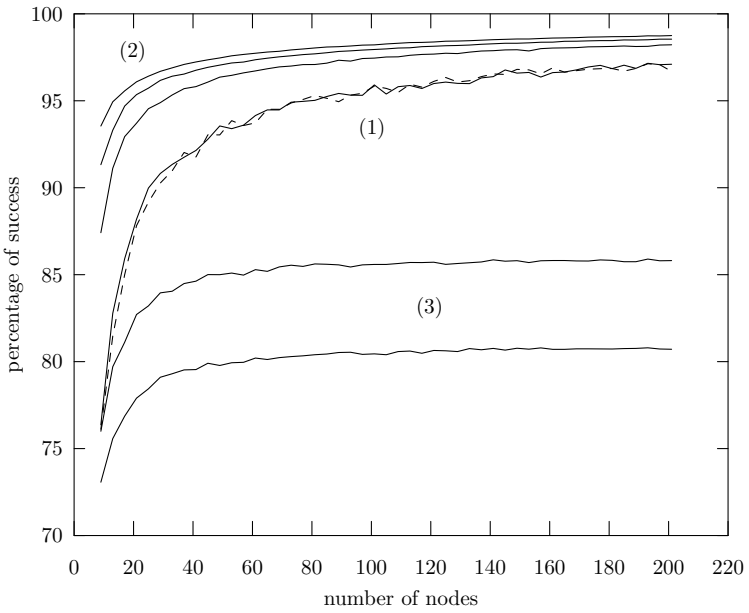

Fig. 4. Density classification task performance as a function of N and K. (1) solid line: $K = 3$, dashed line: $K = 4$, (2) bottom-up, $K = 5, 7, 9$, (3) bottom-up, $K = 8, 6$.

tion, such as changing the state of one node, might be sufficient to drive the system towards the wrong solution.

5.1 The Density Classification Task

Figure 4 shows the network performance for $K \in \{3, 4, 5, 6, 7, 8, 9\}$. All networks were uniform and simply used the majority rule, which, as shown in Section 4.1, is the only common solution when $K \geqslant 3$.

The results show that the performance increases with increasing N. We will later show that it tends to 100% when $N \to +\infty$. Moreover, for odd values of K, RBNs perform better than for even values, and the larger K, the better the performance. This finding confirms that the majority rule works better without ambiguous situations, i.e., when a node does not receive an equal number of 1s and 0s.

Figure 5 shows the average number of iterations a RBN requires to converge to a solution. Networks that need a small number of iterations to solve the density classification task are said to perform better. One can see that the number of iterations required increases with the network's size N. This intuitively makes sense as a larger network needs more time to solve the task because the information must be propagated among the nodes. However, it is worth mentioning that the number of iterations required does not increase linearly with N, as one might expect, but rather follows a logarithmic-like law. This

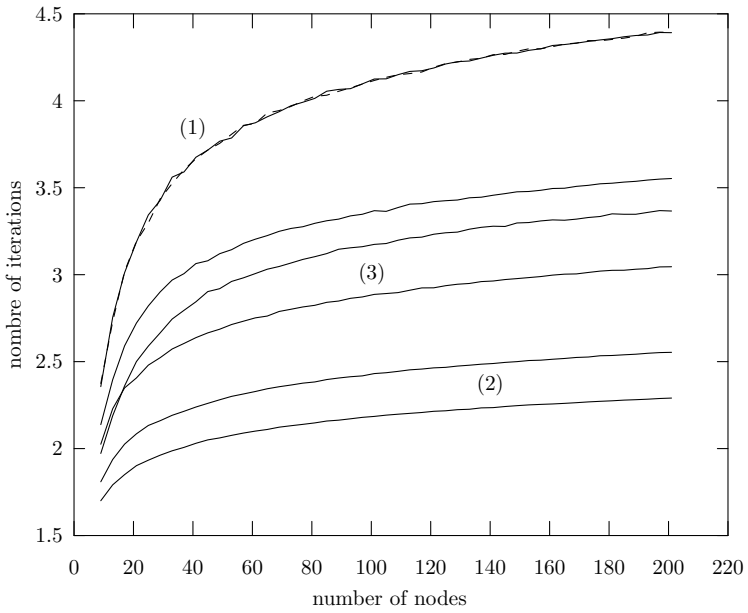

Fig. 5. Average number of iterations required to solve the density classification task as a function of N and K. (1) solid line: $K = 3$, dashed line: $K = 4$, (2) bottom-up, $K = 9, 7$, (3) bottom-up, $K = 5, 8, 6$.

can be explained by the fact that random networks allow for long-distance connections, which help to propagate information faster.

The performance increase with increasing N can be explained by means of Equation 8: although the polynomials are continuous functions, we evaluate them at discrete time steps, i.e., $p_1^t = n_1^t/N$ can only take a finite number of discrete values. For example, let us suppose a network with $N = 10$ and $K = 3$. From a theoretical point of view, if $p_1^t = 6/10$, then $p_1^{t+1} = 0.648$, however, since $p_1^{t+1} \in \{i/N \,|\, i \in [0, N]\}$ the exact value taken by p_1^{t+1} will then depend on the actual network wiring. Performance will therefore get better with increasing N and tends to perfection when $N \to +\infty$. In order to further illustrate this, we generated Figures 6, 7, 8, and 9, that include both the theoretical and the experimental results.

From these four figures we can see that the bigger N, the better the theoretical curves are matched. As expected, we observe large variances for small values of N, which implies that the network may converge toward the wrong solution near the fixed point $p_1^* = 1/2$, since p_1^{t+1} may cross the identity function $Q_K(x) = x$. This explains why the density classification task is solved less successfully for small N.

Moreover, the performance for odd values of K is better because the derivative at the fixed point $p_1^* = 1/2$ tends to be infinite when $K \to N$. Errors in that point are then less likely to make p_1^{t+1} cross the identity function $Q_K(x) = x$, and therefore to drive the network towards the wrong solution. This can easily

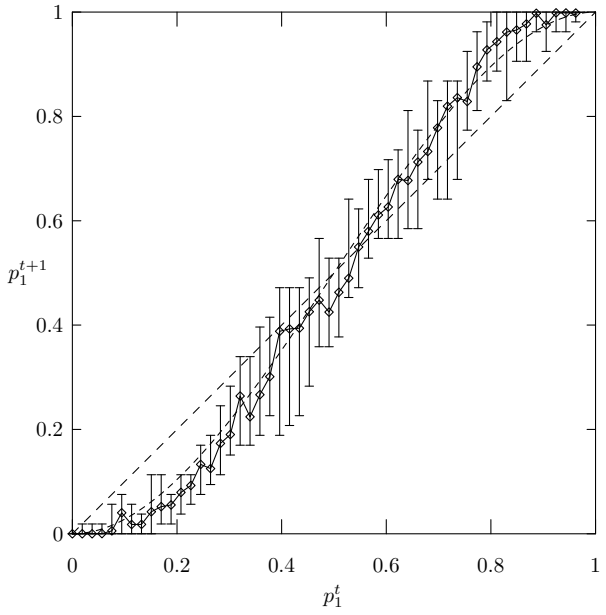

Fig. 6. Density classification task map $p_1^{t+1} = Q_K(p_1^t)$ for $N = 53$ and $K = 3$. Mean (diamond) simulated and theoretical curve (dashed line). The identity function is also shown (dashed line). The bars indicate the maximum and minimum values observed.

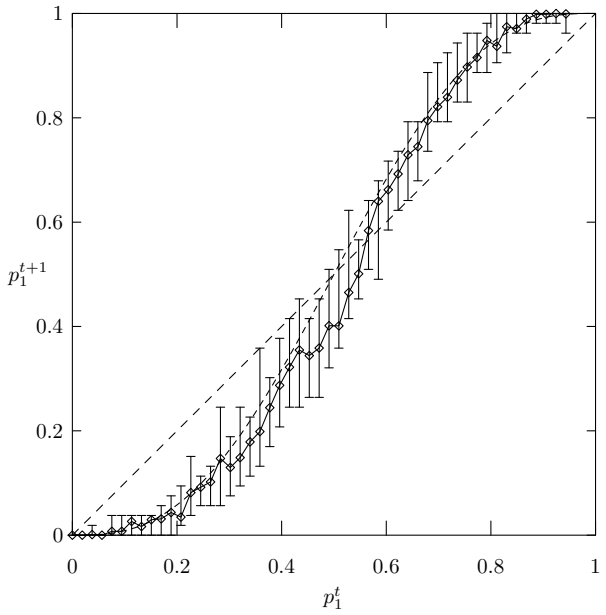

Fig. 7. Density classification task map $p_1^{t+1} = Q_K(p_1^t)$ for $N = 53$ and $K = 5$. Mean (diamond) simulated values and theoretical curve (dashed line). The identity function is also shown (dashed line). The bars indicate the maximum and minimum values observed.

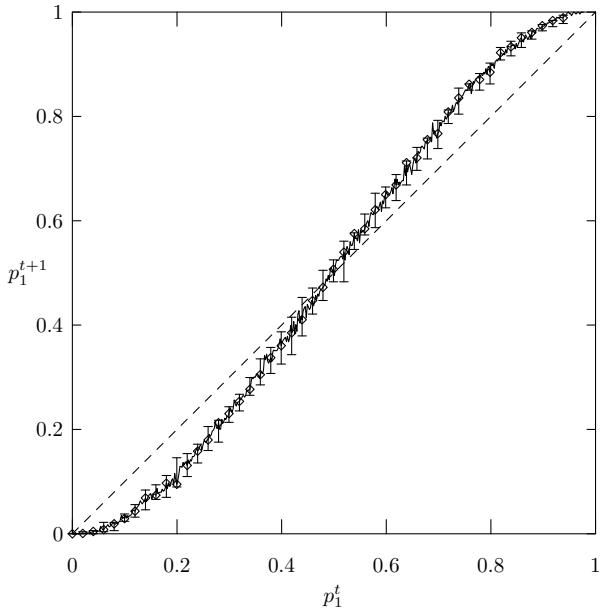

Fig. 8. Density classification task map $p_1^{t+1} = Q_K(p_1^t)$ for $N = 501$ and $K = 3$. Mean (diamond) simulated values and theoretical curve (dashed line). The identity function is also shown (dashed line). The bars indicate the maximum and minimum values observed.

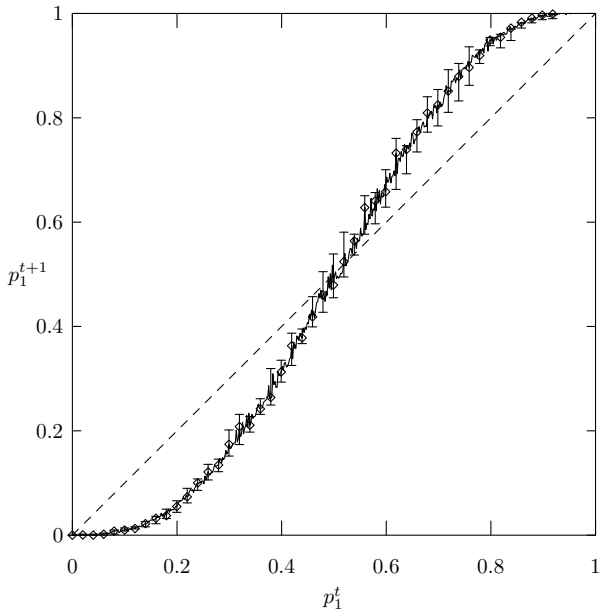

Fig. 9. Density classification task map $p_1^{t+1} = Q_K(p_1^t)$ for $N = 501$ and $K = 5$. Mean (diamond) simulated values and theoretical curve (dashed line). The identity function is also shown (dashed line). The bars indicate the maximum and minimum values observed.

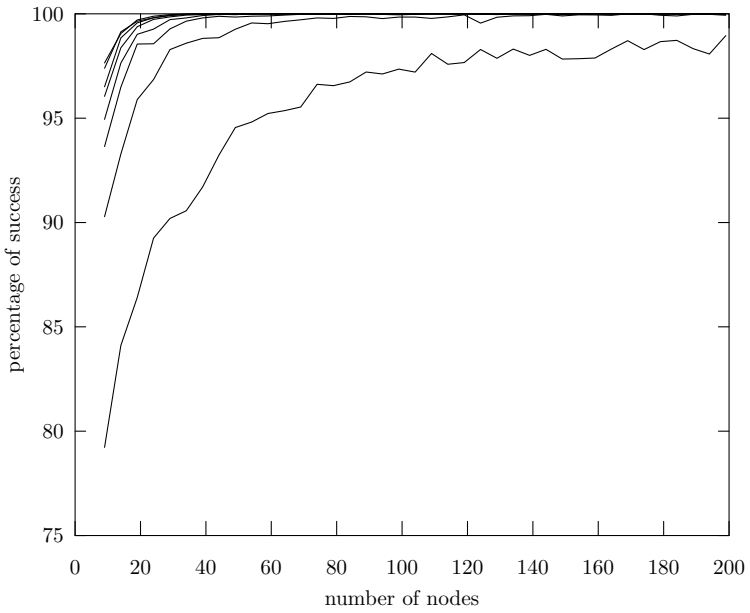

Fig. 10. Synchronization task performance as a function of N and K. Bottom-up, $K = 2, 3, 4, 5, 6, 7, 8, 9$.

be seen if Figures 8 and 9: for $K = 3$, there are at least four points in the neighborhood of $p_1^* = 1/2$ with an extrema that crosses the identity function, while there are only two for $K = 5$. Finally, note that the density classification task is trivially solved in one time step by the majority rule when $K = N$.

5.2 The Synchronization Task

In order to evaluate the performance of RBNs on the synchronization task, we used the following set of parameters (see Algorithm 1): $N_{\min} = 10$, $N_{\max} = 200$, `rep` $= 10,000$ and `nnet` $= 200$. However, the initial configurations were still chosen according to a biased distribution as defined above.

Figure 10 shows the performance of the synchronization task, whereas Figure 11 shows the number of iterations necessary to synchronize the automaton. The network was uniform and all nodes used the γ_K-rule as described in Section 4.2. One can see that, similarly to the density classification task, performance increases with increasing N and K and that it tends to perfection (i.e., 100%) when $N \to +\infty$ or $K \to N$. However, perfect results are already obtained for $N \approx 150$ and $K > 3$. Note that if $K = N$, each node possesses a full view on the entire network state and the task therefore becomes trivial.

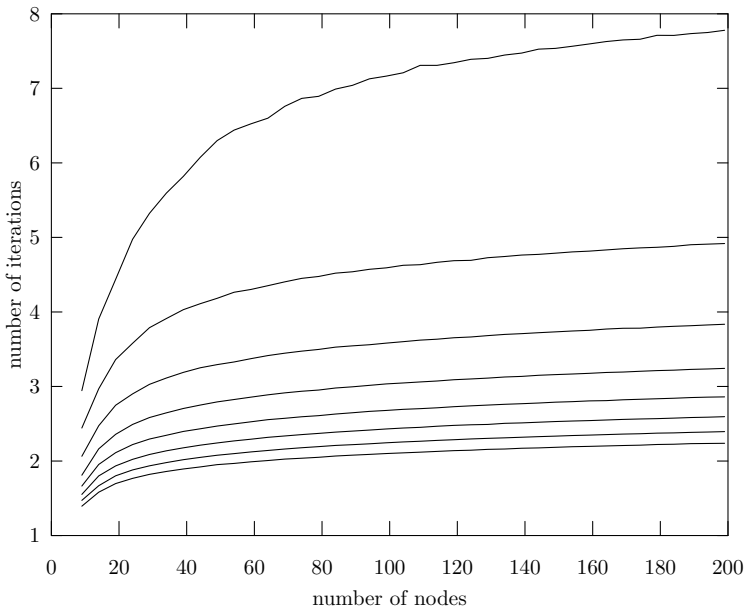

Fig. 11. Average number of iterations required before synchronization occurs as a function of N and K. Top-down, $K = 2, 3, 4, 5, 6, 7, 8, 9$.

5.3 RBNs versus CAs

The performance of CAs for both the density classification and the synchronization task were investigated in various publications (see Section 2.1 and 2.2). We shall compare these results in the present section to those obtained with RBNs and will especially focus on how the information is processed in CAs and RBNs.

In [29], Mitchell et al. presented four different CA rules for the density classification task, namely ϕ_{GKL}, ϕ_{maj}, ϕ_{exp}, and ϕ_{par}, which all used a CA neighborhood of radius $r = 3$, i.e., each cell is connected to its $2r$ nearest neighbors and to itself. ϕ_{maj} is the majority rule, ϕ_{exp} and ϕ_{par} were generated by means of an evolutionary algorithm, and ϕ_{GKL} is a hand-designed rule derived from Gacs, Kurdyumov and Levin's work [17, 19], which is defined as following:

$$s_i^{t+1} = \begin{cases} \text{majority } (s_i^t, s_{i-1}^t, s_{i-3}^t) \text{ if } s_i^t = 0 \\ \text{majority } (s_i^t, s_{i+1}^t, s_{i+3}^t) \text{ if } s_i^t = 1 \end{cases}.$$

ϕ_{exp} [2] uses of a block expansion [29], whereas ϕ_{par} [3] uses a "particle-based" strategy [29]. Furthermore, we considered the hand-written Das-rule [16], the Andre-Bennett-Koza (ABK) rule [5], which was found by means of a genetic

[2] Rule table for a $r = 3$ CA: `0505408305c90101200b0efb94c7cff7`.
[3] Rule table for a $r = 3$ CA: `0504058705000f77037755837bffb77f`.

Type	Rule	$\mathcal{P}_{149,10^4}$	$\mathcal{P}_{599,10^4}$	$\mathcal{P}_{999,10^4}$	Reference
CA	ϕ_{maj}	0.000	0.000	0.000	[29]
CA	ϕ_{exp}	0.652	0.515	0.503	[29]
CA	ϕ_{par}	0.769	0.725	0.714	[29]
CA	ϕ_{GKL}	0.816	0.766	0.757	[29]
CA	Das rule	0.823	0.778	0.764	[16]
CA	ABK rule	0.824	0.764	0.730	[5]
CA	Co-evolution (1)	0.851	0.810	0.795	[23]
CA	Co-evolution (2)	**0.860**	0.802	0.786	[23]
RBN	ϑ_3	0.766	0.771	0.769	Mesot and Teuscher
RBN	ϑ_5	0.823	0.825	0.820	Mesot and Teuscher
RBN	ϑ_7	0.850	**0.848**	**0.852**	Mesot and Teuscher

Table 2
Density classification performance of CAs and RBNs for different network sizes and different rules. The best result in each column is shown in bold.

algorithm, as well as two others rules generated by means of co-evolution [23]. Those rules will be compared to three RBN rules, namely $\vartheta_3, \vartheta_5, \vartheta_7$, where ϑ_i is the majority rule for $K = i$. In order to compare two-states CAs with RBNs, the most natural way is to chose a RBN with a $K = 2r + 1$ neighborhood, however, as we will see, even RBNs with $K < 2r + 1$ perform better than CAs with a r-neighborhood.

In [29], the performance $\mathcal{P}_{N,10^4}$ of a network composed of N nodes is defined as the fraction of correct classifications made over 10^4 initial configurations that are randomly selected from an unbiased distribution, i.e., each bit in the initial configuration is randomly chosen. The expected percentage of 1s in a configuration if therefore 1/2 (see Section 5). For our RBNs, $\mathcal{P}_{N,10^4}$ is averaged over 200 randomly generated networks.

Table 2 summarizes the performance of the density classification task for the different rules. One can see that the ϑ-rules outperform the ϕ-rules since ϑ_3 is as good as the best rule found by the evolutionary algorithm for $N = 149$, and it performs better than ϕ_{GKL} for $N = 599$ or $N = 999$. Furthermore, as mentioned in [16]: "[t]he performance of these rules decreased dramatically for larger N [...]," which is not the case for RBNs, on the contrary, they perform better with increasing N (see Figure 4).

In 1995, Das et al. [15] described the results obtained by four evolved ϕ-rules on the synchronization task. Those rules where used on CAs with radius $r = 3$ and were evaluated by the same procedure as Mitchell et al. [29] used for the

Automaton	Rule	$\mathcal{P}_{149,10^4}$	$\mathcal{P}_{599,10^4}$	$\mathcal{P}_{999,10^4}$	Reference
CA	ϕ_1	0.00	0.00	0.00	[15]
CA	ϕ_2	0.33	0.07	0.03	[15]
CA	ϕ_3	0.57	0.33	0.27	[15]
CA	ϕ_{sync}	1.00	1.00	1.00	[15]
RBN	γ_3	1.00	1.00	1.00	Mesot and Teuscher
RBN	γ_5	1.00	1.00	1.00	Mesot and Teuscher
RBN	γ_7	1.00	1.00	1.00	Mesot and Teuscher

Table 3
Synchronization performance of CAs and RBNs for different network sizes and different rules.

density classification task. Table 3 summarizes the performance of ϕ- and γ-rules for different network sizes. One can see that all γ_K-rules as well as the ϕ_{sync}-rule are able to perfectly solve the synchronization task. We may wonder why a RBN of 149 nodes does not make any error since we have seen in Figure 10 that a network of 200 nodes did not perform perfectly? The difference lies in the fact that performance is essentially evaluated on configurations where the percentage of 1s is around 1/2.

5.4 An Entropy-Based Performance Measure

In order to measure a rule's capability to solve the density classification or the synchronization task, we will introduce the following measure derived from Wuensche's *input entropy* [56]:

$$H(W_1^t \,|\, P_1^t) = \sum_{i=0}^{N} H\left(W_1^t \,|\, P_1^t = i/N\right) P\left(P_1^t = i/N\right), \tag{11}$$

where $H(\cdot)$ is the Shannon entropy [37]. This measure simply computes the conditional entropy of the number of 1s in a node's input, knowing the state of the network at time t, i.e., the percentage of 1s, P_1. Contrary to the measure used by Wuensche, which considers each possible node input, our measure only takes into account the distribution of W_1^t, i.e., the number of 1s in an input. This means that two node input configurations that contain the same number of 1s are considered to be equivalent. Our measure not only allows to qualitatively compare the CA and RBN behavior, but it also gives indications on how difficult it is for a rule to solve a given task and what the average number of required times steps is. For example, if we obtain $H(W_1^t \,|\, P_1^t) = 0$ after a certain amount of time, we know that the network has converged to

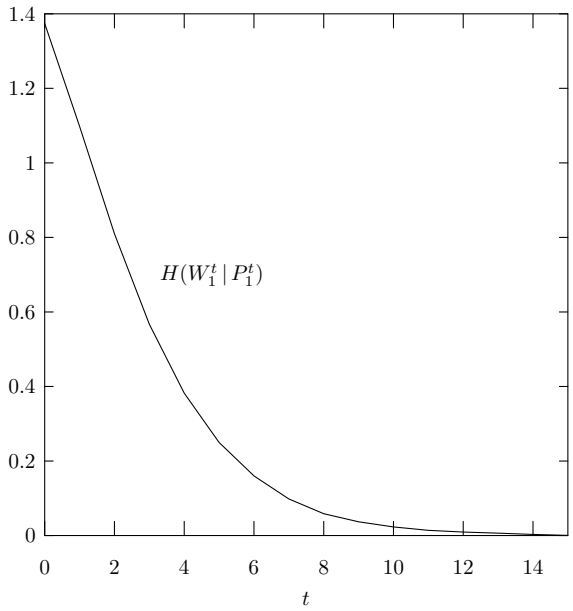

Fig. 12. Theoretical variation of the conditional input entropy, $H(W_1^t \mid P_1^t)$, when a RBN solves the density classification task.

an all 1s or all 0s state. This will therefore provide us with an estimation of how fast a RBN may solve a task.

We have seen in Section 3 that W_1^t follows a binomial law $\mathcal{B}(K, p_1^t)$, where p_1^t is implicitly defined by means of the map $p_1^t = Q_K(p_1^{t-1})$ and by an initial distribution function for P_1^0. For example, Figure 12 shows the variation of the input entropy if we use the map defined by Equation 10 (Section 2.1) under the assumption that the initial configurations are randomly chosen according to a biased distribution, i.e., $P(P_1^0) = \mathcal{U}(0,1)$ [4]. As can be seen, the input entropy quickly decreases towards zero. This means that—at least theoretically—it takes on average less than 15 time steps to settle down in a fixed point when $K = 3$ and the majority rule is used. However, as shown in Figure 5, the actual number of steps required is much lower (the average lies around 4.5 steps). Figures 13 and 14 show the variation of the conditional input entropy for the RBN rule ϑ_7 as well as for the CA rules ϕ_{maj}, ϕ_{exp}, ϕ_{par}, and ϕ_{GKL} ($N = 149$, initial configurations selected using a biased or unbiased distribution respectively, see Section 5.3).

The results suggest the following comments:

- ϑ_7 exactly follows the curve as predicted by our theory.
- All plots start slightly below a value of 2. This value is exactly the same as for a binomial law, which means that at time $t = 0$, no difference exists between RBNs and CAs.

[4] $\mathcal{U}(a, b)$ is the uniform distribution on the interval $[a, b]$.

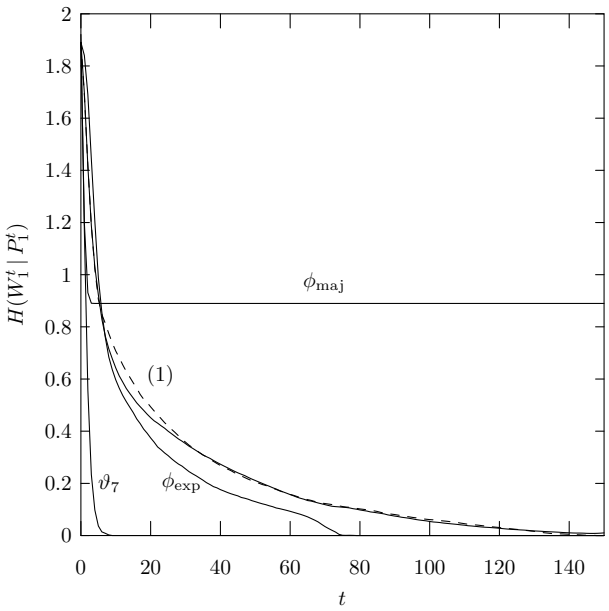

Fig. 13. Variation of the conditional input entropy of different rules for the density classification task. Initial configurations are chosen from a biased distribution and $N = 149$. (1) solid line: ϕ_{GKL}, dashed line: ϕ_{par}.

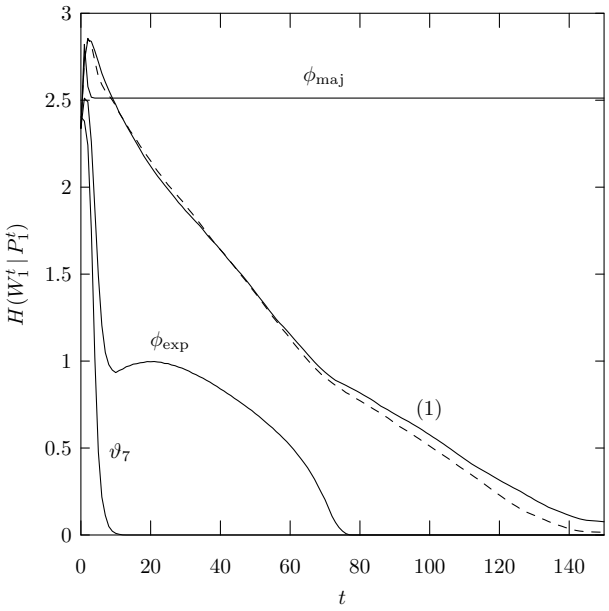

Fig. 14. Variation of the conditional input entropy of different rules for the density classification task. Initial configurations are chosen from an unbiased distribution. $N = 149$. (1) solid line: ϕ_{GKL}, dashed line: ϕ_{par}.

- When $t > 0$, the behaviors of the ϕ-rules are fairly different from the ϑ-rules. As seen in Section 5.4, RBNs follow a simple curve that corresponds to a binomial law. CAs, on the other hand, need more complex behaviors to achieve the same result.
- RBNs with ϑ_7-rule have a very simple, straightforward behavior compared

to CAs. After a very small number of iterations, the input entropy is already less than one bit, which means that most of the nodes basically see the same input.

- The ϕ_{maj}-rule is very similar to the ϑ_7-rule for unbiased distributions. However, because of the CA's connection topology, frozen blocks show up and prevent the automata from converging to a zero input entropy.
- The behavior of the best evolved ϕ_{par}-rule and of the hand-designed ϕ_{GKL}-rule are very similar. Note that ϕ_{GKL} has intentionally been designed to break the inherent symmetry of CAs and thus requires more time to converge.
- The behavior of the ϕ_{exp}-rule is very similar to that of the ϑ_7-rule. However, Figure 14 shows that an increase in complexity is required around $H(W_1^t \mid P_1^t) = 1$ to overcome the emergence of frozen blocks. ϕ_{par} and ϕ_{GKL}, on the other hand, increase the complexity of their behavior from the very beginning in order to somehow compensate for the regularity of the interconnection topology.

Similarly, Figures 15 and 16 show the variation of the conditional input entropy for the synchronization task. The γ_7-rule as well as the three CA rules ϕ_2, ϕ_3 and ϕ_{sync} were used, $N = 149$, and initial configurations were selected using a biased and unbiased distribution respectively (see Section 5.3). The results suggest the following comments:

- γ_7 follows almost exactly the curve as predicted by our theory. A small difference, however, exists when the unbiased distribution is used (Figure 16).
- In the unbiased case, the system converges after less than three iterations, which is very rapid, given that the synchronization is always perfect in that case (see Table 3).
- The input entropy is not fully monotonous (Figure 16). This can be explained by looking at the evolution of the network's state during the first three iterations: after the first iteration, most of the nodes are in state 0 since γ_7 produces a 1 only if the input contains 0s only. The input entropy is thus very low, however, at the second iteration, all nodes which are connected to a 1 will be in state 0 and all others in state 1 since they are connected to 0s only. Thus, a single 1 in the network after the first iteration can generate more than one 0 at the next step and therefore increases the input entropy.
- The non-convergence of ϕ_2 and ϕ_3 after 300 iterations indicates the existence of frozen-blocks, which makes the problem much harder to solve for these rules.

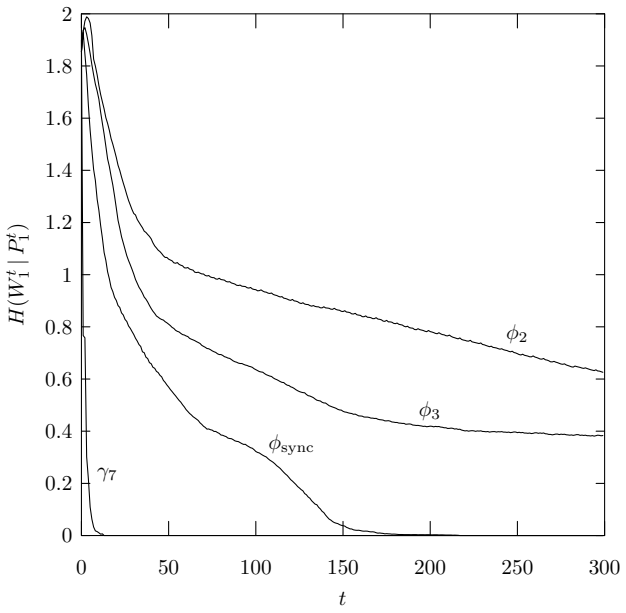

Fig. 15. Variation of the conditional input entropy of different rules for the synchronization task. Initial configurations are chosen from a biased distribution, $N = 149$.

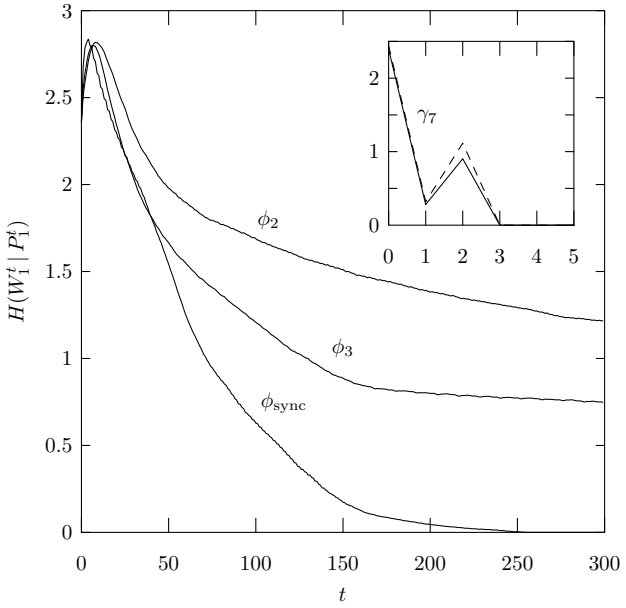

Fig. 16. Variation of the conditional input entropy of different rules for the synchronization task. Initial configurations are chosen from an unbiased distribution, $N = 149$. Subgraph: solid line = simulated γ_7, dashed line = theoretical γ_7.

6 Non-Standard Architectures and Topologies

In order to improve the computational power of CAs, various people proposed to introduce non-uniformity among the rules. Although this approach sacrifices one of the main advantages of CAs, namely, to have simple and identical

processing elements, it allows to find better solutions [38, 41, 46]. These solutions often rely on a particular structure where the nodes' rules must be carefully chosen. With RBNs, however, this is not the case: as we have seen in Section 4, when $K = 5$ for example, a single map can represent $14,000$ rules, which are strictly equivalent. We can thus easily build non-uniform RBNs by randomly choosing one rule for each node instead of using the same rule in all nodes.

CAs and RBNs can be considered as two extreme classes of networks that are not really observed in Nature. From the point of view of the interconnection topology, CAs are simply too regular, whereas random networks require an equal probability for each connection to be established among all nodes, which is in most cases subjected to physical limitations since long-distance connections are more costly than short-distance connections. Many recent studies have instead confirmed that an important number of networks observed in Nature belong to a class called *small-world* networks (see for example [1,8,49,54]). These networks may be considered as an intermediate class between CAs and RBNs, since, if we gradually transform a CA into a RBN in a certain manner, we can obtain small-world networks.

Different types of small-world networks exist [3], here we shall however only consider the networks of Watts and Strogatz as described in [54]. In their paper they state for the density classification task that "[...] a simple *majority-rule* running on a small-world graph can outperform all known human and genetic algorithm-generated rules running on a ring lattice." However, they do not give the amount of randomness a network should contain in order to outperform CAs on the density task. Moreover, since we also studied the synchronization task, it would be interesting to see if small-world networks using the γ-rule perform better than CAs on this task as well. We performed several experiments with networks built in the following way:

(1) Start with a ring lattice, which might be considered as a special one-dimensional CA with a $r = 1$ neighborhood, but without self-connections.
(2) Rewire the connections with probability ρ as described in [54].

Therefore, if $\rho = 0$, the initial automaton is left unchanged, if $\rho = 1$, it is transformed into an automaton with a random topology. Note that, although the connectivity K was 2 for each node at the beginning, each node can potentially have a different connectivity at the end. The average of $K = 2$ connections per node is, however, not affected by this random rewiring algorithm.

Figures 17 and 18 show the percentage of success and the number of iterations required for solving the density classification and the synchronization task as a function of ρ, i.e., the amount of randomness in the network. The simulations were performed for a network size of $N = 149$ nodes, the rules

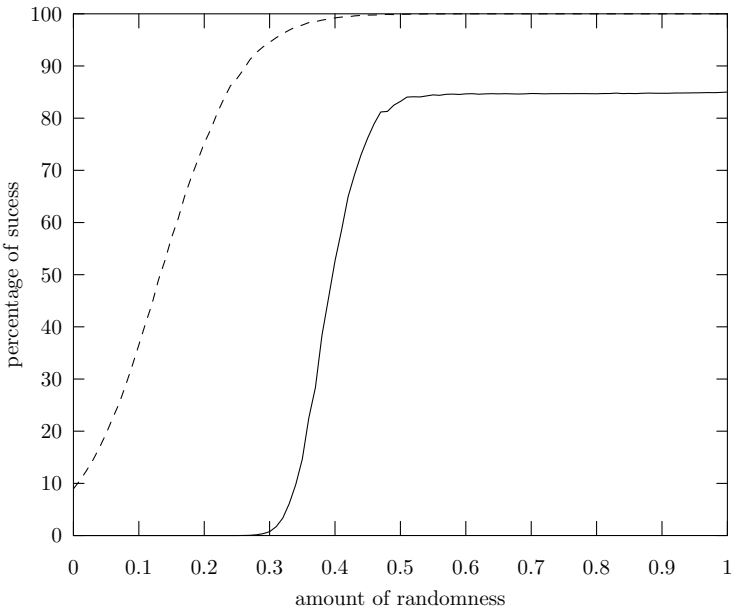

Fig. 17. Percentage of success of small-world networks for the density classification (solid line) and the synchronization task (dashed line) as a function of the amount of randomness ρ. $N = 149$, majority rule (density) and γ-rule (synchronization).

used were the majority rule for the density classification and the γ-rule for the synchronization task.

The results can be summarized as following: For both tasks, the percentage of success reaches its maximal value around $\rho = 0.5$ (see Figure 17). Whereas the number of iterations for the synchronization task reaches its maximum around $\rho = 0.3$ (Figure 18) already, the density classification task requires significantly less iterations on a random topology. The reason for this can be found in the corresponding maps (Figures 2 and 3): the synchronization map does not possess stable fixed points, which makes it less sensitive to additional connections between the already existing clusters.

We can conclude that a relatively small amount of randomness already greatly helps to improve performance for solving our two tasks. However, especially for the density classification task, a randomly interconnected network helps to significantly improve convergence. Despite the extreme simplicity of our random Boolean networks used, the results fit into the global picture of recent work on complex network synchronization using coupled oscillators (see for example [32, 34]), although more works is certainly needed in that area.

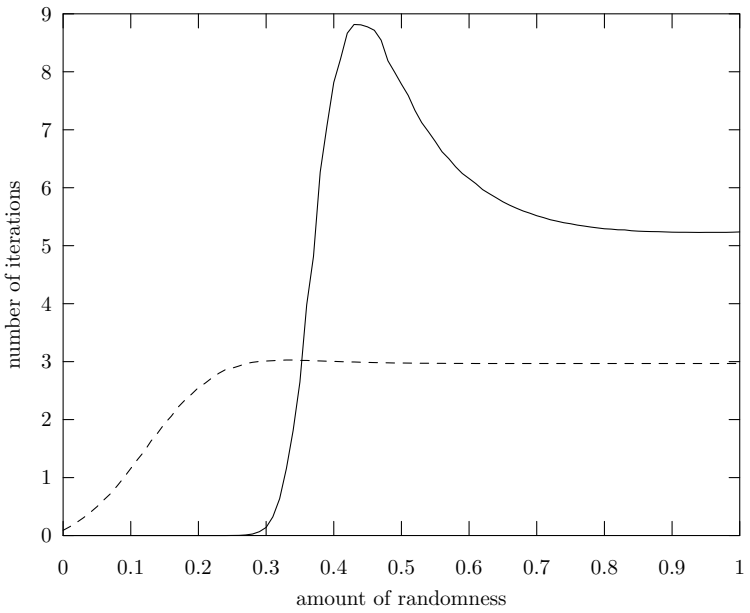

Fig. 18. Number of iterations required by small-world networks for solving the density classification (solid line) and the synchronization task (dashed line) as a function of the amount of randomness ρ. $N = 149$, majority rule (density) and γ-rule (synchronization).

7 Conclusions

In this paper we have first derived a recursive equation which gives the probability of a node to be in state 1 at time $t + 1$ for random Boolean networks. Based on that equation, we were then able to analytically find the local node rules from the global network behavior for our two comparative tasks. Our main findings are the following:

(1) No $K = 2$ random Boolean network can perfectly, i.e., for all initial configurations, solve the density classification task. The majority rule is the only perfect rule for $K = 3$ networks, whereas many rules exist for $K > 3$ networks.
(2) The two symmetrical γ_K-rules are the best rules for solving the synchronization task.
(3) Random Boolean networks outperform cellular automata on both the density classification and on the synchronization task.
(4) Random Boolean networks also outperform networks with a small-world topology on the density classification task, but their performance is equivalent for the synchronization task.
(5) The rules obtained for both our tasks are highly scalable, i.e., the larger the network, the better they perform. This is not the case for cellular automata.

The reason why random Boolean networks perform better on both the synchronization and on the density classification tasks is the following: due to the random wiring, each node of a random Boolean network has—from a statistical point of view—an unbiased view on the automata's global state since the distribution of 1s in the node inputs of each node is the same as in the global network state. On the other hand, cellular automata have a biased view because of their local neighborhood, which makes it more difficult for them to solve global tasks.

Another advantage of random Boolean networks over CAs is that they allow the network to be rewired at any time, without affecting the global performance. This property is especially interesting if interconnections are likely to fail. One would then only have to re-establish a new connection to a randomly chosen alternative node. This property could be very interesting for large scale distributed systems. Although not explicitly tested in this work, the existence of many equivalent rules for certain values of K tend to show that RBNs may exhibit great robustness to node and interconnection failures.

One might of course ask whether the current approach might be applied to similar global tasks and whether it might be generalized to non-uniform random Boolean networks, alternative topologies, or asynchronously updating nodes. Our current work concentrates exactly on these questions. The goal is to extend the theory to asynchronously updating and non-uniform random Boolean networks and to other, more useful tasks. Asynchrony not only allows to remove the sole global clock signal necessary to synchronously update the cells, which often represents a constraint for hardware realizations, but also yields in further potentially interesting properties. Asynchronous automata attracted much interest in the past few years, although it has been proved independently by Capcarrère [13] and Nehaniv [33] that any n-state synchronous automata can be emulated by a particular $3n^2$-state asynchronous automata. In their 1984 experimental study, Ingerson and Buvel [22] already explored the question of how much of the interesting behavior of cellular automata comes from the synchronous modeling. They concluded—based on the visual appearance of the evolution patterns over time—that the synchronous assumption is not essential to the study of cellular automata and that certain irrelevant structures may appear from the synchronous update of the cells. However, one of the main arguments against purely synchronous automata as a tool for modeling has always been the lack of biological plausibility [48], although a purely asynchronous behavior is certainly not biologically plausible either. But there are several other issues of interest in asynchronous models: Bersini and Detours [9] suggested that asynchrony might induce stability, but also forces evolution to find more robust solutions, as Rohlfshagen and Di Paolo have shown for the case of rhythmic asynchronous random Boolean networks [35].

A further topic of interest is also the extension of the theory to S-state random networks, which are likely to have interesting properties and would represent a generalization of S-state CAs. The formalization of asynchronously updating and non-uniform random Boolean or S-state networks and the ability to derive their local rules for a larger class of global tasks would certainly represent a major step in that field of research, where the local node rules are commonly either hand-designed or evolved so far.

Acknowledgments

The authors are grateful to Jonas Buchli for his helpful comments and discussions. This work was supported in part by the Swiss National Science Foundation under grant PBEL2–104420.

References

[1] R. Albert and A.-L. Barabasi. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97, January 2002.

[2] J. T. Allanson. Some properties of randomly connected neural nets. In C. Cherry, editor, *Proceedings of the 3rd London Symposium on Information Theory*, pages 303–313, Butterworths, London, 1956.

[3] L. A. N. Amaral, A. Scala, M. Barthélémy, and H. E. Stanley. Classes of small-world networks. *Proceedings of the National Academy of Sciences (PNAS)*, 97(21):11149–11152, 2000.

[4] S. I. Amari. Characteristics of randomly connected threshold-element networks and network systems. *Proceedings of the IEEE*, 59(1):35–47, January 1971.

[5] D. Andre, F. B. Bennett III, and J. R. Koza. Evolution of intricate long-distance communication signals in cellular automata using genetic programming. In C. G. Langton and K. Shimohara, editors, *Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*, volume 1, pages 16–18, Cambridge, MA, 1996. MIT Press.

[6] M. Andrecut and M. K. Ali. Chaos in a simple boolean network. *International Journal of Modern Physics B*, 15(1):17–23, 2001.

[7] W. R. Ashby, H. von Forster, and C. C. Walker. Instability of pulse activity in a net with threshold. *Nature*, 196:561, 1966.

[8] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, October 15 1999.

[9] H. Bersini and V. Detours. Asynchrony induces stability in cellular automata based models. In R. A. Brooks and P. Maes, editors, *Proceedings of the Artificial Life IV conference*, pages 382–387, Cambridge, MA, 1994. MIT Press.

[10] M. Capcarrère and M. Sipper. Necessary conditions for density classification by cellular automata. *Physical Review E*, 6403(3):6113–6117, 2001.

[11] M. Capcarrère, M. Tomassini, and M. Sipper. A $r = 1$, two-state cellular automata that classifies density. *Physical Review Letters*, 77(24):4969–4971, 1996.

[12] M. Capcarrère, M. Tomassini, A. Tettamanzi, and M. Sipper. A statistical study of a class of cellular evolutionary algorithms. *Evolutionary Computation*, 7(3):255–274, 1999.

[13] M. S. Capcarrère. *Cellular Automata and Other Cellular Systems: Design & Evolution*. PhD thesis, PhD Thesis No 2541, Swiss Federal Institute of Technology, Lausanne, 2002.

[14] M. S. Capcarrère. Evolution of asynchronous cellular automata. In J. J. Merelo Guervós, A. Panagiotis, and H.-G. Beyer, editors, *Parallel Problem Solving from Nature*, volume 2439 of *Lecture Notes in Computer Science*, pages 903–912, Berlin, Heidelberg, 2002. Springer-Verlag.

[15] R. Das, J. P. Crutchfield, M. Mitchell, and J. E. Hanson. Evolving globally synchronized cellular automata. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 336–343, San Francisco, CA, 1995. Morgan Kaufmann.

[16] R. Das, M. Mitchell, and J. P. Crutchfield. A genetic algorithm discovers particle-based computation in cellular automata. In Y. Davidor, H.-P. Schwefel, and R. Manner, editors, *Proceedings of the Third Parallel Problem-Solving From Nature Conference (PPSN III)*, volume 866 of *Lecture Notes in Computer Science*, pages 344–353, Berlin, Heidelberg, 1994. Springer-Verlag.

[17] P. Gonzaga de Sá and C. Maes. The Gacs-Kurdyumov-Levin automaton revisited. *Journal of Statistical Physics*, 67(3/4):507–522, 1992.

[18] S. F. Edwards and P. W. Anderson. Theory of spin glasses. *Journal of Physics F*, 5:965, 1975.

[19] P. Gacs, G. L. Kurdyumov, and L. A. Levin. One-dimensional uniform arrays that wash out finite islands. *Problemy Peredachi Informatsii*, 14:92–98, 1978.

[20] C. Gershenson. Classification of random boolean networks. In Standish et al. [47], pages 1–8.

[21] W. Hordijk. The structure of the synchonizing-CA landscape. Technical Report 96-10-078, Santa Fe Institute, Santa Fe, NM (USA), 1996.

[22] T. E. Ingerson and R. L. Buvel. Structure in asynchronous cellular automata. *Physica D*, 10(1–2):59–68, January 1984.

[23] H. Juillé and J. B. Pollack. Coevolving the "ideal" trainer: Application to the discovery of cellular automata rules. In J. R. Koza, W. Banzhaf, K. Chellapilla, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. L. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 519–527, San Francisco, CA, 1998. Morgan Kaufmann.

[24] S. A. Kauffman. Metabolic stability and epigenesis in randomly connected genetic nets. *Journal of Theoretical Biology*, 22:437–467, 1968.

[25] S. A. Kauffman. Emergent properties in random complex automata. *Physica D*, 10(1–2):145–156, January 1984.

[26] S. A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, New York; Oxford, 1993.

[27] M. Land and R. K. Belew. No perfect two-state cellular automata for density classification exists. *Physical Review Letters*, 74(25):5148–5150, 1995.

[28] M. T. Matache and J. Heidel. Random boolean network model exhibiting deterministic chaos. *Physical Review E*, 69:056214, 2004.

[29] M. Mitchell, J. P. Crutchfield, and R. Das. Evolving cellular automata with genetic algorithms: A review of recent work. In *Proceedings of the First International Conference on Evolutionary Computation and its Applications (EvCA'96)*. Russian Academy of Sciences, 1996.

[30] M. Mitchell, J. P. Crutchfield, and P. T. Hraber. Evolving cellular automata to peform computations: Mechanisms and impediments. *Physica D*, 75:361–391, 1994.

[31] M. Mitchell, Peter T. Hraber, and J. P. Crutchfield. Revisiting the edge of chaos: Evolving cellular automata to perform computations. *Complex Systems*, 7(2):89–130, 1993.

[32] A. E. Motter, C. S. Zhou, and J. Kurths. Emhancing complex-network synchronization. *Europhysics Letters*, 69(3):334–340, 2005.

[33] C. L. Nehaniv. Evolution in asynchronous cellular automata. In Standish et al. [47], pages 65–73.

[34] T. Nishikawa, A. E. Motter, Y.-C. Lai, and F. C. Hoppensteadt. Heterogenity in oscillator networks: Are smaller worlds easier to synchronize. *Physical Review Letters*, 91(1):014101, 2003.

[35] P. Rohlfshagen and E. A. Di Paolo. The circular topology of rhythm in asynchronous random boolean networks. *BioSystems*, 73(2):141–152, February 2004.

[36] L. I. Rozonoér. Random logical nets I. *Automation and Remote Control*, 5:773–781, 1969. Translation of Avtomatika i Telemekhanika.

[37] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, pages 379–423, 623–656, July, October 1948.

[38] M. Sipper. Non-uniform cellular automata: Evolution in rule space and formation of complex structures. In R. A. Brooks and P. Maes, editors, *Artificial Life IV*, pages 394–399, Cambridge, MA, 1994. MIT Press.

[39] M. Sipper. Quasi-uniform computation-universal cellular automata. In F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, editors, 3^{rd} *European Conference on Artificial Life*, Lecture Notes in Computer Science, pages 544–554, Berlin, Heidelberg, 1995. Springer-Verlag.

[40] M. Sipper. Studying artificial life using a simple, general cellular model. *Artificial Life*, 2(1):1–35, 1995.

[41] M. Sipper. Co-evolving non-uniform cellular automata to perform computations. *Physica D*, 92:193–208, 1996.

[42] M. Sipper. *Evolution of Parallel Cellular Machines: The Cellular Programming Approach*. Springer-Verlag, Heidelberg, 1997.

[43] M. Sipper, M. S. Capcarrère, and E. Ronald. A simple cellular automaton that solves the density and ordering problems. *International Journal of Modern Physics C*, 9(7):899–902, October 1998.

[44] M. Sipper, M. Goeke, D. Mange, A. Stauffer, E. Sanchez, and M. Tomassini. The firefly machine: Online evolware. In *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, pages 181–186, Piscataway, NJ, USA, 1997. IEEE Press.

[45] M. Sipper and E. Ruppin. Co-evolving architectures for cellular machines. *Physica D*, 99:428–441, 1997.

[46] M. Sipper and M. Tomassini. Computation in artificially evolved, non-uniform cellular automata. *Theoretical Computer Science*, 217:81–98, 1999.

[47] R. K. Standish, M. A. Bedau, and H. A. Abbass, editors. *Artificial Life VIII. Proceedings of the Eight International Conference on Artificial Life*. Complex Adaptive Systems Series. A Bradford Book, MIT Press, Cambridge, MA, 2003.

[48] W. R. Stark and W. H. Hughes. Asynchronous, irregular automata nets: The path not taken. *BioSystems*, 55(1-3):107–117, February 2000.

[49] S. H. Strogatz. Exploring complex networks. *Nature*, 410:268–276, 2001.

[50] C. Teuscher and M. S. Capcarrère. On fireflies, cellular systems, and evolware. In A. M. Tyrrell, P. C. Haddow, and J. Torresen, editors, *Evolvable Systems: From Biology to Hardware. Proceedings of the* 5^{th} *International Conference (ICES2003)*, volume 2606 of *Lecture Notes in Computer Science*, pages 1–12, Berlin, Heidelberg, 2003. Springer-Verlag.

[51] T. Toffoli and N. Margolus. *Cellular Automata Machines*. MIT Press, Cambridge, MA, 1987.

[52] M. Tomassini and M. Venzi. Evolving robust asynchronous CAs for the density task. *Complex Systems*, 13(3):185–204, 2002.

[53] J. von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana, Illinois, 1966.

[54] D. J. Watts and S. H. Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393:440–442, 1998.

[55] S. Wolfram. Cellular automata as models of complexity. *Nature*, 311:419–424, December 4 1984.

[56] A. Wuensche. Classifying cellular automata automatically. *Complexity*, 4(3):47–66, February 1999.