Theoretical Computer Science 609 (2016) 118–128

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

PSPACE-completeness of majority automata networks

Eric Goles [a], Pedro Montealegre [b,1], Ville Salo [c,2], Ilkka Törmä [d,*,3]

[a] Facultad de Ingeniería y Ciencias, Universidad Adolfo Ibáñez, Santiago, Chile
[b] Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, Orléans, France
[c] Centro de Modelamiento Matemático, Universidad de Chile, Santiago, Chile
[d] University of Turku, TUCS – Turku Centre for Computer Science, Finland

ARTICLE INFO

Article history:
Received 19 January 2015
Accepted 12 September 2015
Available online 30 September 2015
Communicated by J. Karhumäki

Keywords:
Boolean network
Majority network
Prediction problem
PSPACE

ABSTRACT

We study the dynamics of majority automata networks when the vertices are updated according to a block sequential updating scheme. In particular, we show that the complexity of the problem of predicting an eventual state change in some vertex, given an initial configuration, is **PSPACE**-complete.

1. Introduction

A *threshold network* is a dynamical system over a connected undirected graph, where at each vertex is assigned a *state* that evolves at discrete time steps accordingly to a *vertex threshold function*, that depends on the current state of the vertex and the states of their neighbors in the graph. In this paper we study a particular case of threshold networks, called *majority networks*, where at each time step, the vertices take the state that the majority of their neighbors have. Such dynamical systems have been used to model a variety of biological, physical and social phenomena [3,8,2,4]. However, those systems are not always precisely modeled with a synchronous updating of each vertex, which raises the need for considering different ways of updating the network.

An *updating scheme* is a total preorder over the set of vertices, such that at each time step, vertices that are first in this scheme evolve before the others. Updating schemes are classified in three groups: *synchronous*, *sequential* and *block-sequential*. A synchronous updating scheme means that every vertex evolves in parallel. Sequential updating schemes are the other extreme: no two vertices are updated at the same time. The block-sequential updating schemes are an intermediate situation, where the vertices are partitioned into collections called *blocks*, and the vertices of each block update at the same time.

* Corresponding author.
E-mail addresses: eric.chacc@uai.cl (E. Goles), pedro.montealegre@etu.univ-orleans.fr (P. Montealegre), vsalo@dim.uchile.cl (V. Salo), iatorm@utu.fi (I. Törmä).
[1] Research supported by CONICYT-Becas Chile – 72130083.
[2] Research partially supported by FONDECYT 1140090, ECOS C12E05 and Basal project PFB-03, Centro de Modelamiento Matemático, UMI 2807 CNRS, Universidad de Chile.
[3] Research supported by the Academy of Finland Grant 131558.

http://dx.doi.org/10.1016/j.tcs.2015.09.014
0304-3975/© 2015 Elsevier B.V. All rights reserved.

A natural problem in automata networks is *prediction*: given an initial configuration and an updating scheme, to predict the future states. This problem has been studied at least in [6,7,10,11]. One possible strategy is to simulate the evolution of each vertex step by step; since the automata network is finite, the dynamics will eventually enter a loop, and the simulation strategy will result in a complete description of the evolution of the network. A straightforward follow-up question is if this solution is *efficient*, i.e. if there exist better solutions, or if this strategy outputs in *reasonable* time. To evaluate this, the measure chosen is the computational complexity of the problem. In [5] it is shown that, for threshold networks in general, the simulation strategy is indeed efficient for both synchronous and sequential updating schemes. This was shown by proving that the simulation strategy always stops in a number of steps that is polynomial in the size of the graph, and then the prediction problem is in the class **P**. Further, it is shown that for block sequential updating schemes and the majority networks, the problem is unlikely to be efficiently solved (it is **NP**-hard), leaving the exact complexity classification open.

In this paper we show that in majority networks the prediction problem, restricted to block-sequential updating schemes, is **PSPACE**-Complete. We prove this by showing that majority networks can simulate iterated Boolean circuits, whose prediction problem is easily seen to be **PSPACE**-Complete. Later, we show that this result remains true even when we limit the block sequential updating schemes to have only blocks of constant size (only a constant number of vertices can be updated at the same time), or a constant number of blocks (the number of groups of vertices that are not updated at the same time is constant). In this context *constant* means independent of the number of vertices of the network. Finally, we show how this result is also generalizable to other decision problems, as well as to a more general form of the majority rule.

2. Preliminaries

Let $G = (V, E)$ be a simple connected undirected graph, where V is a finite set of vertices and E the set of edges. An *automata network* is a tuple $\mathcal{A} = (G, (f_v)_{v\in V})$, where $f_v : \{0,1\}^V \to \{0,1\}$ is the *vertex function* associated to the vertex v. Here, $\{0,1\}$ is the set of *states*, and vertices in state 1 are called *active* while vertices in state 0 are *inactive*. We say that the vertex functions are the *rule* of $\mathcal{A}$. The set $\{0,1\}^V$ is called the set of *configurations*. We sometimes refer to the sets V or E, or the vertex functions f_v, without having explicitly introduced them, but in these cases, they should be clear from the context.

Let $N(v) \subset V$ be the set of neighbors of vertex $v \in V$, with $v \notin N(v)$. We are interested in the vertex function, called the *majority function*, defined by

$$f_v(x) = \begin{cases} 1, & \text{if } \sum_{u\in N(v)} x_j > \frac{|N(v)|}{2}, \\ 0, & \text{if } \sum_{u\in N(v)} x_j \le \frac{|N(v)|}{2} \end{cases}$$

for all configurations $x \in \{0,1\}^V$. This means that a vertex will become active if the strict majority of its neighbors are active, and will become inactive otherwise. In our proofs, the tie-breaking rule is never activated, so it could be chosen arbitrarily.

An *updating scheme* of the automata network $\mathcal{A}$ is a function $S : V \to \{1, \ldots, |V|\}$ such that if $u, v \in V$ and $S(u) < S(v)$, then the state of u is updated before v, and if $S(u) = S(v)$, then the vertices u and v are updated at the same time. When $S(u) = S(v)$ for all $u, v \in V$, so that all vertices are updated at the same time, we have the *synchronous* updating scheme. When S is injective, that is, no two vertices are updated at the same time, we have a *sequential* updating scheme. In our formalism, every updating scheme S is *block sequential*, meaning that the vertex set is partitioned into the subsets $V_k = S^{-1}(k) \subset V$ for $k \in \{0, \ldots, |V|\}$, such that the sets are updated one after the other, in the order $V_1, V_2, \ldots, V_{|V|}$, and elements inside each are updated synchronously.

More precisely, a network $\mathcal{A}$ and an updating scheme S define a *global transition function* $F_S : \{0,1\}^V \to \{0,1\}^V$ as follows. For each $k \in \{1, \ldots, |V|\}$, let $F_k : \{0,1\}^V \to \{0,1\}^V$ be the function that updates the vertices in turn k according to S:

$$F_k(x)_v = \begin{cases} f_v(x), & \text{if } S(v) = k, \\ x_v, & \text{otherwise.} \end{cases}$$

Then we define $F_S = F_n \circ F_{n-1} \circ \cdots \circ F_1$. For $t \ge 0$, when the updating scheme S is clear from the context, we sometimes denote $F_S^t(x) = x(t)$ for brevity. The iterated application of the global transition function on a given initial configuration generates a dynamic on the set of all configurations. For instance, Fig. 1 shows three updating schemes for a graph of nine vertices and the majority rule. Notice that for the same initial configuration we can obtain different dynamics.

Given an automata network $\mathcal{A}$ and an updating scheme S, the *trajectory* of a configuration $x \in \{0,1\}^V$ is the infinite sequence $T_S(x) = (F_S^t(x))_{t\in\mathbb{N}} = (x, x(1), x(2), \ldots)$. When there is no ambiguity, we omit the subindex S. The notation $y \in T_S(x)$ means that $y = x(t)$ for some $t \in \mathbb{N}$. Since the set of configurations is finite (its size is $2^{|V|}$), every trajectory is eventually periodic, so there exist $t \in \mathbb{N}$ and $p > 0$ such that $x(t) = x(t+p)$. We say that the trajectory of x *enters a limit cycle of period* p if p satisfies the above and is minimal, and then the set $\{x(t), x(t+1), \ldots, x(t+p-1)\} \subset \{0,1\}^V$ is the *limit cycle* of x. A cycle of period 1 is a *fixed point*. The set of configurations in a limit cycle is called an *attractor*.

We define the *transient length* $\tau_S(x) \in \mathbb{N}$ of a configuration x under the updating scheme S as the number of steps required to reach from x, for the first time, a configuration in the attractor, and we define the transient length of the automata network $\mathcal{A}$ under the updating scheme S as the greatest of these values:

$$\tau_S(\mathcal{A}) = \max\{\tau_S(x) : x \in \{0,1\}^V\}$$

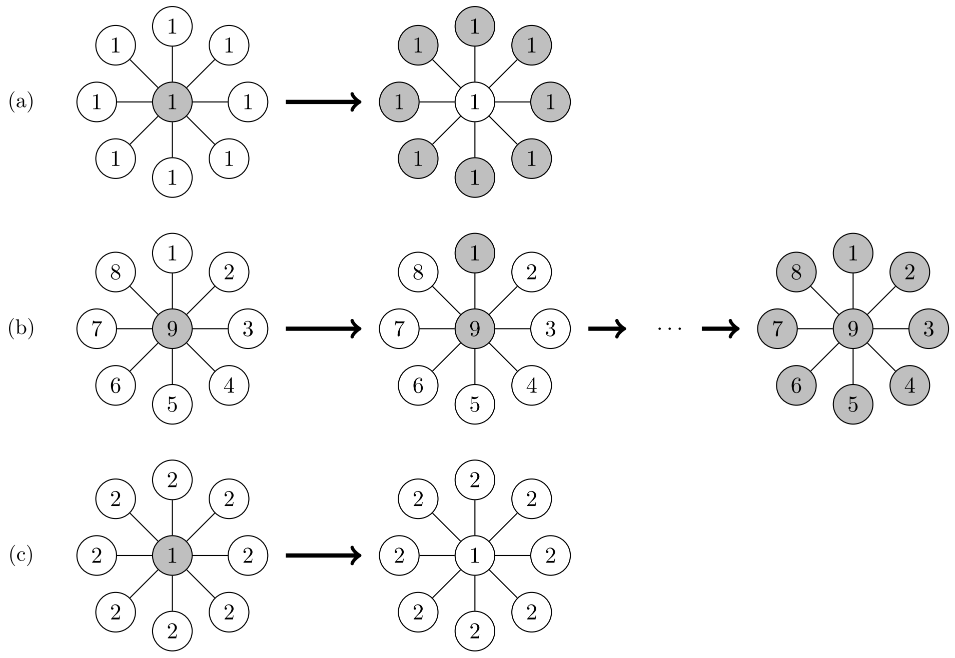

Fig. 1. Examples of dynamics in a majority network on the same initial configuration under different updating schemes: synchronous (a), sequential (b) and block sequential (c). Gray vertices are active while white ones are inactive. The number of a vertex is its value in the updating scheme.

Suppose that we would like to make predictions about the attractor associated to a configuration x of a majority network $\mathcal{A}$ with updating scheme S. Clearly a solution is to simulate the evolution of each vertex until we reach a limit cycle. However, this process can *a priori* take as many steps as there are configurations, or $2^{|V|}$. A straightforward question is whether more efficient solutions exist, like some algebraic or algorithmic properties that would allow us to make predictions on the size of the attractors, or which vertices will eventually change their initial states. For a family $\mathcal{S}$ of updating schemes, we define the following decision problem, called the *one vertex prediction problem*:

$\mathcal{S}$-Majority: Given a majority network $\mathcal{A} = (G, (f_v)_{v\in V})$, updating scheme $S \in \mathcal{S}$, a configuration $x \in \{0,1\}^V$ of $\mathcal{A}$, and a vertex $v \in V(G)$ that is initially inactive ($x_v = 0$), does there exist $y \in T_S(x)$ such that $y_v = 1$?

The family $\mathcal{S}$ may contain the synchronous, sequential or block sequential updating schemes. For each case we will have an associated decision problem: Syn-Majority, Seq-Majority, and Bseq-Majority, respectively.

The computational complexity of a decision problem is defined as the amount of resources (usually time or space) required to find an answer, as a function of the input size. Classical complexity theory considers the following fundamental classes: **P**, the class of problems solvable by a deterministic Turing machine in polynomial time; **NP**, the class of problems solvable by a nondeterministic Turing machine in polynomial time; and **PSPACE**, the class of problems solvable by a deterministic Turing machine that uses polynomial space. It is well known that **P** $\subset$ **NP** $\subset$ **PSPACE**.[4] Informally, **P** is the class of problems with a *feasible* solution in terms of the execution time; **NP** is the class of problems where it is feasible to verify a given solution; and **PSPACE** is the class of problems with a feasible solution in terms of the space required to solve the problem.

It is a well-known conjecture that **P** $\neq$ **NP**, and if so, then there exist problems whose solution is feasible to verify, while actually finding a solution is infeasible. The problems in **NP** which are the most likely to not belong to **P** are the **NP**-complete problems, to which any other problem in **NP** can be reduced by a polynomial time reduction. Thousands of **NP**-complete problems are known, the best-known example probably being the *Boolean satisfiability problem* Sat [1]. It is also conjectured that **NP** $\neq$ **PSPACE**, and similarly to **NP**-Complete problems, the problems in **PSPACE** that are the most likely to not belong to **NP** or **P** are the **PSPACE**-complete problems, to which any other problem in **PSPACE** can be reduced in polynomial time. One can also define a notion of completeness for **P**, and we say a problem is **P**-complete if every problem in **P** can be reduced to it by a parallel algorithm in polylogarithmic time.

[4] By $A \subset B$, we denote a not necessarily strict inclusion.

Since we explicitly use Turing machines with certain properties in our reductions, we give our definition for them. In this article, a *deterministic Turing machine* is a seven-tuple $M = (Q, \Gamma, \Sigma, \delta, B, q_i, q_f)$, containing the *state set* Q, *tape alphabet* Γ, *input alphabet* $\Sigma \subset \Gamma$, *transition function* $\delta : Q \times \Gamma \to Q \times \Gamma \times \{-1, 0, 1\}$, *blank symbol* $B \in \Gamma \setminus \Sigma$, and *initial and final states* $q_i, q_f \in Q$. It operates on a right-infinite tape, and is initialized at the leftmost tape cell. We say that M is *linear bounded*, if there exists $K > 0$ such that on every input $w \in \Sigma^*$, the machine visits at most $K|w|$ distinct tape cells. The *linear bounded prediction problem*, denoted Linear-Bounded, is the problem of determining, given a linear bounded deterministic Turing machine M, an input $w \in \Sigma^*$, and a padding of length $K|w|$, whether M accepts w in space $K|w|$. It is easy to show that this problem is **PSPACE**-complete.

A *Boolean circuit* is a directed acyclic graph C whose vertices that are not sources are labeled with either $\wedge$ or $\vee$, or possibly $\neg$ if their in-degree is exactly one. The source vertices of C are called its *inputs*, the sinks are called *outputs*, and the other vertices are called *gates*. The circuit is *monotone* is it contains no gates with label $\neg$. If C has n inputs and m outputs, it computes a function $C : \{0, 1\}^n \to \{0, 1\}^m$ in the obvious way. For each gate of a circuit, its *layer* is the length of the shortest path from an input to the gate. The *iterated (monotone) Boolean circuit problem* is the problem of determining, given a (monotone) Boolean circuit $C : \{0, 1\}^n \to \{0, 1\}^n$, an input string $x \in \{0, 1\}^n$ and an index $i \in \{1, \dots, n\}$, whether there exists $t \in \mathbb{N}$ such that $C^t(x)_i = 1$. It is denoted Iter-Circuit (Iter-Mon-Circuit, respectively).

2.1. Previous results

Threshold automata networks, called also *neural networks*, have been widely studied [6,7,10,11]. In [7], Goles et al. give a characterization of the attractors and transients of such networks through the use of a monotone operator, analogous to the *spin glass energy* [10]. It is shown that for the synchronous updating scheme, when the adjacency matrix A of the underlying graph is symmetric (equivalently, the graph is undirected, which always holds in our case), then the attractors are only fixed points or cycles of period two. Further, if the diagonal elements of the matrix are non-negative (it has no self-loops of negative weight, which again holds in our case), then the sequential updating scheme admits only fixed points. If A is the adjacency matrix of an automata network of n vertices, the transient lengths are bounded by $\mathcal{O}(n^2)$, no matter what updating scheme is used. In terms of complexity, these results imply that the problems Syn-Majority and Seq-Majority are in **P**, since the dynamics of the majority rule reaches a limit cycle of length at most 2 in $\mathcal{O}(n^2)$ steps, so that direct simulation leads to a polynomial time algorithm. In the same paper, it was shown that both problems are **P**-complete, which means that they are not likely to be efficiently parallelizable [9].

For more general updating schemes, that is, block-sequential updating schemes, it is shown in [5] that it is possible to construct majority automata networks with limit cycles of any period, and using these structures in an appropriate way, it is shown that majority automata networks with block-sequential updating schemes admit limit cycles with super-polynomial period in the size of the network. The possibility of large periods suggests that, unlike in the synchronous and sequential cases, it is not possible to have a monotone operator associated to a majority rule for block-sequential updating schemes. Moreover, in [5] it is shown that Bseq-majority is **NP**-hard, and it is conjectured that this problem is in fact **PSPACE**-complete.

2.2. Contributions

Our main contribution is the proof of the conjecture proposed in [5], showing that Bseq-Majority is **PSPACE**-complete, and more specifically, our proof shows that this remains true when we restrict the block sequential updating scheme to have blocks of constant size, or a constant number of blocks. The complete proof is given in Section 3.

In Section 4 we prove that several other decision problems related to majority networks are also **PSPACE**-Complete. Indeed, we show that if we restrict the block-sequential updating scheme to have a constant number of blocks, the corresponding decision problem is still **PSPACE**-Complete. On the other hand, in Bseq-Majority one ask for changes in a single vertex, at some step, given an initial configuration. We show that it is still **PSPACE**-complete to ask for changes if the initial configuration is not a part of the input, if we ask for changes for infinitely many steps, or if we ask for changes in every vertex of the network at the same time.

Finally, in Section 5 we show that our results apply to a sort of *generalized majority*, where this time a vertex becomes active not if the majority of their neighbors are active, but if a portion p of the number of neighbors, where p is a fixed constant in $(0, 1)$. We finish our paper with some conclusions and remarks.

3. Bseq-Majority is PSPACE-complete

We begin this section by remarking two properties of majority networks. First, Lemma 1 states that majority networks can simulate monotone Boolean circuits. Then, Lemma 2 states that majority networks can exhibit large limit cycles using blocks of size 2. The gadgets shown in the proofs are essentially found already in [5].

Lemma 1. *For every monotone Boolean circuit $C : \{0, 1\}^n \to \{0, 1\}^m$, there is a majority network $\mathcal{M}$ defined over a graph G of size polynomial in $|C|$ with vertices $v_1, \dots, v_n, w_1, \dots, w_m \in V(G)$, and with a sequential update scheme S such that for all $I \in \{0, 1\}^n$, if*

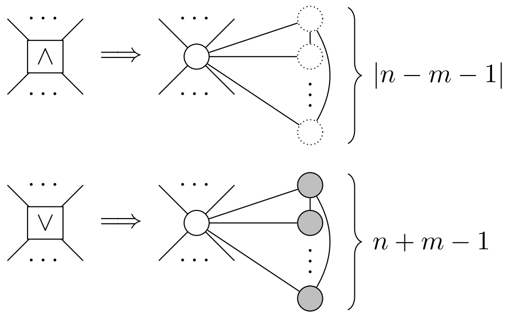

Fig. 2. The gadgets of the logic gates (the gates have indegree n and outdegree m). Inputs are on the top. The dotted nodes have value 0 if $n-m-1 \geq 0$, and 1 otherwise.

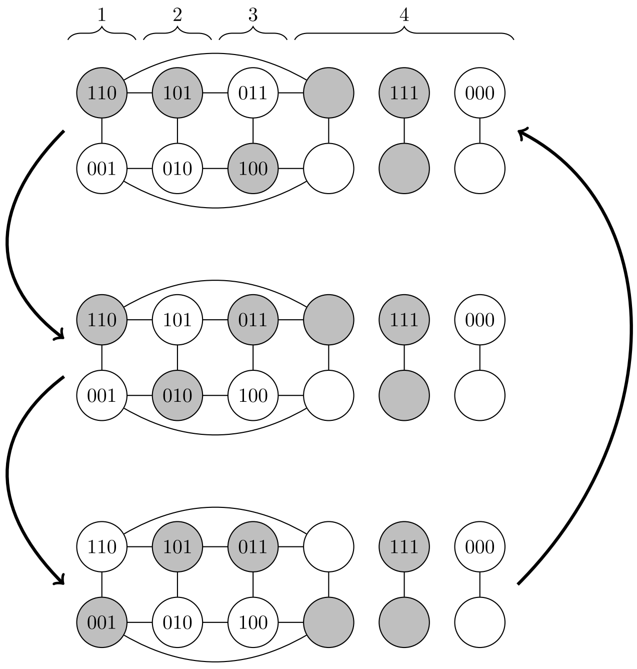

Fig. 3. The clock gadget and its dynamics. Gray vertices are active while white ones are inactive. The updating scheme is given by the numbering in the top.

we set $x_{v_i} = I_i$ for all $i \in \{1, \dots, n\}$ and $x_v = 0$ for all other $v \in V(G)$, then $F_S(x)_{w_j} = C(x)_j$ for all $j \in \{1, \dots, m\}$. Moreover, if every logic gate of C has degree at most d, then every node of G has degree at most $2d - 1$.

Proof. We can assume that the inputs of C have out-degree exactly one. The circuit C is transformed into G as follows. To the inputs and outputs are assigned the vertices v_i and w_j, respectively. The AND-gates and OR-gates are replaced by the gadgets shown in Fig. 2. For the update scheme we can choose any ordering where the inputs of a logic gate are updated before the gate itself, except that the inputs are updated last. $\square$

Lemma 2. *There exists a majority automata network $\mathcal{M}$ over a graph H of degree 3 containing the vertices v_s for $s \in \{0, 1\}^3$ and four additional vertices, a block sequential update scheme S, and an initial configuration x such that for all $s \in \{0, 1\}^3$ and $t \in \mathbb{N}$ we have $x(t)_{v_s} = s_{t \bmod 3}$.*

Proof. This graph is shown in Fig. 3. The updating scheme is given by the numbering in the figure, and an easy simulation confirms the claim. $\square$

Lemma 3. *The problem* Iter-Mon-Circuit *is* ***PSPACE****-complete, even when restricted to circuits of degree 4 (in-degree and out-degree 2).*

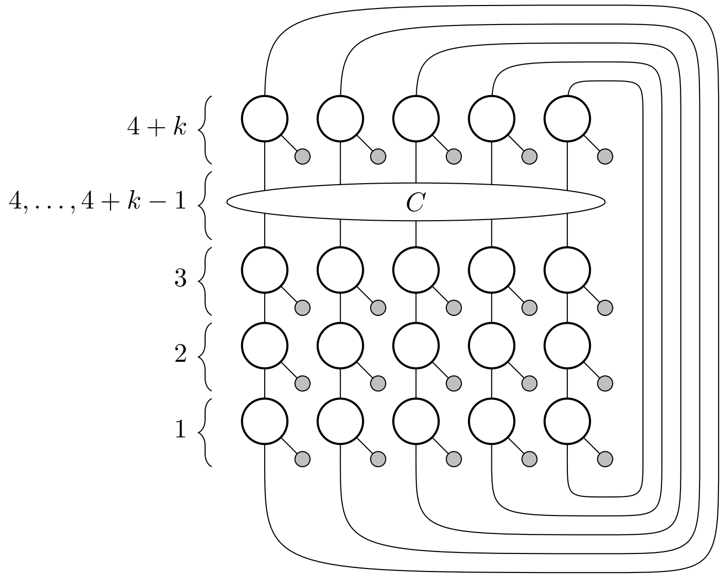

Fig. 4. The clocked network, shown here with input length 5. The small gray vertices will always be active, and the circuit C computes from top to bottom. The number of update blocks in the circuit is k. Note how the upper edges reappear cyclically at the bottom.

Proof. First, it is clear that Iter-Circuit is **PSPACE**-complete, since the **PSPACE**-complete problem Linear-Bounded easily reduces to it. Namely, given a linear bounded Turing machine $M = (Q, \Gamma, \Sigma, \delta, B, q_i, q_f)$, an input w of length ℓ and a padding of length $K|w|$, we construct a Boolean circuit with $K\ell \log_2 |\Gamma|$ inputs for the tape cells, $\log_2 |Q|$ inputs for the internal state, $\log_2 K\ell$ inputs for the location of the read–write head on the tape, one extra input for signaling the halting state, and some intermediate gates that calculate the next computation step.

To simulate a Boolean circuit by a monotone one, we replace each input v by two inputs v_+ and v_-, so that v_+ is true if and only if v is true, and v_- is true if and only if v is false; similarly, we replace each gate by two gates, a gate computing the value of the original gate, and another gate computing the negation. A $\neg$-gate is simulated by swapping the two inputs, and we use De Morgan's laws to obtain the substitutions for the monotone gates:

$$u \wedge v \mapsto (u_+ \wedge v_+, u_- \vee v_-), \quad u \vee v \mapsto (u_+ \vee v_+, u_- \wedge v_-)$$

It is clear that the v_+-gates of the monotone circuit behave exactly the gates of the original one. Finally, using the construction in [9, Theorem 6.2.3], we can transform the monotone circuit into one where the in-degree and out-degree of every gate is bounded by 2. □

Now we define a more powerful model of majority dynamics on Boolean graphs, and show that the related problem is **PSPACE**-complete.

Definition 4. A *clocked automata network* is an automata network graph G, whose every node $v \in V(G)$ is associated with an element $c(v) \in \{U, 1, 0\}^3$, called its *clock*. The elements U, 1 and 0 are just labels that refer to 'update normally,' 'switch to 1,' and 'switch to 0,' respectively. The dynamics of a clocked Boolean graph is the same as that of the underlying automata network, except that on a timestep $t \in \mathbb{N}$, if the symbol $c(v)_{t \bmod 3}$ of the clock is 1 or 0, instead of taking the majority function of its neighborhood, the vertex v simply assumes the respective state.

The decision problem associated to clocked Boolean graphs with majority dynamics and block-sequential updates is denoted Clock-Majority.

Lemma 5. *The problem* Clock-Majority *is* ***PSPACE****-Complete, even when restricted to graphs of degree* 7 *with a sequential update scheme.*

Proof. We reduce Iter-Mon-Circuit to the clocked majority problem. See Fig. 4. Updating is done in the order of the numbering on the left, and within a level, the ordering of nodes can be arbitrary, except that the Boolean circuit C (represented by the ellipse and transformed into a Boolean graph with k update blocks using Lemma 1) is updated in its natural order, that is, inputs of a gate are updated before the gate. We can assume that C has degree 4, so that the clocked network has degree at most 7 by Lemma 1 and the construction.

The small gray vertices have the clock $(1, 1, 1)$, so they will always be active. Other vertices on levels 1 and $4+k$ have clock $(0, 0, U)$, the nodes on levels $3, 4, \ldots, 4+k-1$ have clock $(U, 0, 0)$ and those on level 2 have clock $(0, U, 0)$. It is easy

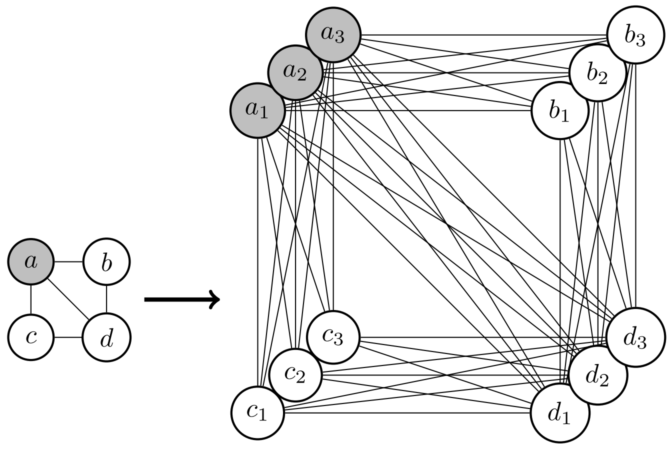

Fig. 5. The amplification operation applied to a simple majority network with $k = 1$.

to check that in three steps, if we begin with the top and bottom rows containing an input vector $x \in \{0, 1\}^n$ and the other rows being empty, we return to the same configuration except with the top and bottom rows containing the string $C(x)$. □

We show now that majority networks can simulate clocked ones. First, we prove the following lemma, which states that using a linear blow-up in the number of vertices, we can make the dynamics of a majority network robust to adding a constant number of vertices to it.

Lemma 6. *Let $\mathcal{A} = (G, (f_v)_{v \in V(G)})$ be an automata network with a block sequential update scheme S, where every the degree of every vertex in G is odd and bounded by $d \in \mathbb{N}$, and let $k \in \mathbb{N}$. Then there exists an automata network $\mathcal{A}^k = (G^k, (\tilde{f}_v)_{v \in V(G^k)})$ of size $(2k+1)|V(G)|$ and degree at most $(2k+1)d$ with a block sequential update scheme S^k, and a $(2k+1)$-to-one function $\phi : V(G^k) \to V(G)$ such that the following holds. For all $x \in \{0, 1\}^{V(G)}$, let $y \in \{0, 1\}^{V(G^k)}$ be such that $y_v = x_{\phi(v)}$ for all $v \in V(G^k)$. Then for all $v \in V(G^k)$ and all $t \geq 0$, we have $F^t_S(x)_{\phi(v)} = F^t_{S^k}(y)_v$, even if G^k is an induced subgraph of any graph H of an automata network $\mathcal{H}$ with $|N_H(v) \setminus G^k| \leq k$ for all $v \in V(G^k)$.*

The automata network $\mathcal{A}^k$ is called the *k-amplification* of $\mathcal{A}$.

Proof. The amplification $\mathcal{A}^k$ has vertex set $V(G^k) = V(G) \times \{1, 2, \ldots, 2k+1\}$ and edge set

$$E(G^k) = \{\{(v, i), (w, j)\} \mid \{v, w\} \in E(G), i, j \in \{1, 2, \ldots, 2k+1\}\}.$$

The function ϕ and the updating scheme S are simply defined by $\phi((v, i)) = v$ and $S^k((v, i)) = S(v)$ for all $(v, i) \in V(G^k)$. The claim easily follows. □

See Fig. 5 for an example of the amplification operation.

Let $\mathcal{BS}_k$ be the family of block sequential updating schemes with block size smaller than k.

Theorem 7. *$\mathcal{BS}_8$-Majority is **PSPACE**-complete, even when restricted to graphs of degree at most* 23.

Proof. We reduce Clock-Majority to said problem. Given a clocked majority automata network $\mathcal{A}$ with underlying graph G of degree $d \in \mathbb{N}$, we construct a majority automata network $\mathcal{B}$ from it as follows. First, we attach a $\lceil d_v/2 \rceil$-fold amplified copy $\mathcal{M}^{\lceil d_v/2 \rceil}$ of the clock network $\mathcal{M}$ of Lemma 2 to every vertex $v \in V(G)$, where $d_v \in \mathbb{N}$ is the degree of v. Denote this subgraph by $\mathcal{C}_v$. Now, $\mathcal{C}_v$ contains at least $d_v + 1$ copies of every vertex v_s in $\mathcal{M}$, any of which we can attach to v without affecting the behavior of $\mathcal{C}_v$. Suppose that the clock of v is $c(v) \in \{0, 1, U\}^3$, and let $c^{\downarrow}(v)$ and $c^{\uparrow}(v)$ be the elements of $\{0, 1\}^3$ obtained by replacing every U in $c(v)$ by 0 or 1, respectively. Then we attach $d_v + 1$ copies of the vertices $c^{\downarrow}(v)$ and $c^{\uparrow}(v)$ in $\mathcal{C}_v$ to v; if $c^{\downarrow}(v) = c^{\uparrow}(v)$, we attach them only once. For example, if the clock of v is $(1, U, 0)$, then we can attach it to $d_v + 1$ copies of the vertex 100, and $d_v + 1$ copies of 110. It is clear from the properties of the clock gadget that the resulting majority automata network $\mathcal{B}$ behaves like the clocked majority automata network $\mathcal{A}$. Furthermore, if the degree of G is bounded by d and its update scheme is sequential, and if we update the clocks one at a time, then the graph of $\mathcal{B}$ has degree at most $3d + 2$ and block size at most $d + 1$. The claim follows, since we can choose $d = 7$. □

Since $\mathcal{BS}_k$ is a subset of the whole family of block sequential updating schemes, we obtain as a corollary that Bseq-Majority is **PSPACE**-Complete.

Corollary 8. Bseq-Majority *is* ***PSPACE****-complete.*

We have shown that the problem Bseq-Majority is **PSPACE**-complete when the maximum block size is constant with respect to the size of the majority network. In the following, we will show that if we limit the total number of blocks instead of their size, then the problem is still **PSPACE**-Complete.

We start by showing that in our circuit simulations, it is enough to deal with circuits of depth 1, that is, circuits with just two layers, input and output. Consider the following decision problem.

1-Depth-Iter-Mon-Circuit: Given a depth-1 monotone Boolean circuit $C : \{0,1\}^n \to \{0,1\}^n$, an input string $x \in \{0,1\}^n$ and an index $i \in \{1,\dots,n\}$, decide if whether there exists $t \in \mathbb{N}$ such that $C^t(x)_i = 1$.

We show that this problem is **PSPACE**-complete.

Lemma 9. 1-Depth-Iter-Mon-Circuit *is* ***PSPACE****-complete, even when restricted to circuits of degree at most* 4.

Proof. This problem is clearly in **PSPACE**. We will show that we can reduce Iter-Mon-Circuit to 1-Depth-Iter-Mon-Circuit in polynomial time, in a way that preserves the maximum in-degrees and out-degrees of vertices. Let (C, x, i) an instance of Iter-Mon-Circuit, where $C : \{0,1\}^n \to \{0,1\}^n$ is a circuit of size N and depth D, $x \in \{0,1\}^n$ is an input string and $i \in \{1,\dots,n\}$ an index. We can assume that C is *synchronous*, that is, every input of a gate in layer ℓ comes from layer $\ell - 1$, and also that each layer of C has the same number $n = N/D$ of gates. Indeed, we can obtain a synchronous monotone circuit from any monotone circuit by increasing the number of gates by a polynomial factor [9]. For each $\ell \in \{0,\dots,D\}$, we enumerate the gates in layer ℓ as $\{\ell n + 1, \ell n + 2, \dots, (\ell+1)n\}$.

We build from C a monotone Boolean circuit $\overline{C}$ with Dn inputs, Dn outputs and depth 1 as follows. First, we enumerate both the input and output gates of $\overline{C}$ as $\{1,\dots,Dn\}$, the latter of which have the same types ($\wedge$ or $\vee$) as their counterparts in C. Then, for all $\ell \in \{0,\dots,D-1\}$ and $k, k' \in \{1,\dots,n\}$, if there is a wire in C from the gate numbered $\ell n + k$ to the gate numbered $(\ell+1)n + k'$, then we add a wire in $\overline{C}$ from the input numbered $\ell n + k$ to the output numbered $(\ell+1)n + k'$ (Fig. 6).

Let $(x, 0^{(D-1)n})$ be the vector of size Dn where the first n components are x and the rest are zeros, and let C_ℓ be the circuit of n inputs and n outputs and depth ℓ, obtained by taking the first ℓ layers of C. It follows directly from the construction of $\overline{C}$ that $\overline{C}^k(x, 0^{(D-1)n}) = (0^{kn}, C^k(x), 0^{(D-(k+1))n})$ for every $1 \le k \le D-1$. Moreover, $\overline{C}^D(x, 0^{(D-1)n}) = (C(x), 0^{(D-1)n})$. It follows that for all $t \ge 0$, we have $\overline{C}^{tD}(x, 0^{(D-1)n})_i = C^t(x)_i$, and $\overline{C}^{tD+\ell}(x, 0^{(D-1)n})_i = 0$ if $0 < \ell < D$. Clearly both $\overline{C}$ and $x0^{(D-1)n}$ can be constructed in a time that is polynomial in the size of C, and the maximum in-degree and out-degree of $\overline{C}$ are exactly those of C, which finishes the proof. □

Let $\mathcal{BN}_k$ the family of block sequential updating schemes with a number of blocks smaller than k.

Corollary 10. *The problem* $\mathcal{BN}_8$-Majority *is* ***PSPACE****-complete, even when restricted to graphs of degree at most* 23.

Proof. This result is proved similarly to Theorem 7, with the following modifications. First, we can assume by Lemma 9 that the circuit C has depth 1, so that it adds no extra levels to the network. Second, the clock gadgets are not updated one by one, but in parallel, as are the vertices of every level. Then the total number of update blocks is the number of levels, plus 4 (from the clocks), or 8 in total. □

4. Other decision problems: the complexity of the majority rule

As the title of this paper suggest, we propose that the 'correct' measure of the complexity of a class of majority automata networks is the complexity of the prediction problem $\mathcal{S}$-Majority restricted to that class, or in other words, the problem of predicting the state changes in some particular vertex of a given network. In this section we give some evidence for this statement by showing that there are many other natural prediction problems that Bseq-Majority can be reduced to, showing that they are also **PSPACE**-complete. Below are three examples of such problems.

$\mathcal{S}$-Eventual-Prediction: Given a majority network $\mathcal{A} = (G, (f_v)_{v \in V})$, an updating scheme $S \in \mathcal{S}$, an initial configuration $x \in \{0,1\}^V$ of $\mathcal{A}$, and an initially inactive vertex $v \in V(G)$, does there exists $t_0 > 0$ such that $x(t)_v = 1$ for all $t \ge t_0$?

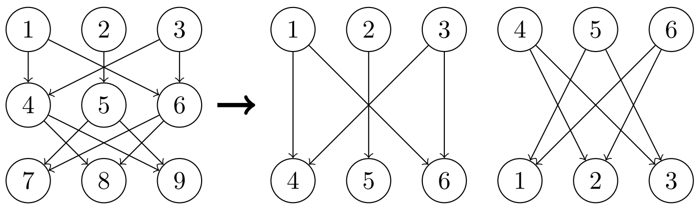

Fig. 6. An example of transforming a circuit C of depth 2 into $\bar{C}$. Note that the outputs of $\bar{C}$ are not in the correct order in the figure.

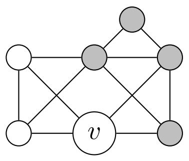

Fig. 7. The gadget of Theorem 11.

Theorem 11. Bseq-Eventual-Prediction *is **PSPACE**-complete.*

Proof. The Bseq-Majority problem can be reduced to this problem by attaching the gadget of Fig. 7 to the vertex $v \in V(G)$ of the Bseq-Majority problem (the central vertex in the figure), and considering either of the new inactive vertices. They become active as soon as v does, and then stay active forever. The problem is still in **PSPACE**, since we can just simulate the graph until we reach a period, which can be easily detected. □

$\mathcal{S}$-Conditional-Prediction: Given a majority network $\mathcal{A} = (G, (f_v)_{v\in V})$, an updating scheme $S \in \mathcal{S}$, a subset $W \subset V$ and a partial configuration $y \in \{0,1\}^W$, and an initially inactive vertex $v \in W$, does there exist a configuration $x \in \{0,1\}^V$ with $x|_W = y$ such that $x(t)_v = 1$ for some $t > 0$?

Theorem 12. Bseq-Conditional-Prediction *is **PSPACE**-Complete.*

Proof. This problem is clearly at least as hard as Bseq-Majority, and it is in **PSPACE** since we can simply enumerate all possibilities for the values of the vertices $V \setminus W$ in polynomial space, and run the simulations. □

$\mathcal{S}$-Full-Prediction: Given a majority network $\mathcal{A} = (G, (f_v)_{v\in V})$, an updating scheme $S \in \mathcal{S}$, and an initial configuration $x \in \{0,1\}^V$, does there exist a time $t > 0$ such that $x(t)_v = 1$ for all $v \in V$?

Theorem 13. Bseq-Full-Prediction *is **PSPACE**-Complete.*

Proof. This problem is also clearly in **PSPACE**, and the Bseq-Majority problem can be reduced to it as follows. Suppose we are given a majority network $\mathcal{A}$ with underlying graph G with maximum degree bounded by an odd number $d \geq 3$, and the special vertex $v \in V(G)$. Enumerate the vertices of G as $v_1, \ldots, v_k$ such that $v_1 = v$. For each $i \in \{1, \ldots, k\}$, add to G two new complete graphs, K_i^0 with inactive vertices and K_i^1 with active vertices, of respective sizes d and $3d - 1$. Introduce an edge from each vertex of K_i^0 to v_i, to each vertex of $K_{i+1 \bmod k}^0$, and to each vertex in a subset of K_i^1 of size $2d$ (in the case $i = 1$, to every vertex of K_i^1). From any d vertices of K_i^1, introduce new edges to v_i. See Fig. 8 for a visualization of the construction. Call this new majority network $\mathcal{B}$ with underlying graph H. We extend the update scheme of $\mathcal{A}$ to $\mathcal{B}$ so that the original nodes are updated first, then the new ones in any order.

Now, every vertex of G has received exactly $2d$ new neighbors, half of which are initially inactive. It is clear that the vertices of each K_i^1 will always stay active, since they have $3d - 2$ active neighbors in K_i^1 itself, and at most $d + 1$ other neighbors. If every vertex of K_1^0 stays inactive, then for $i \neq 1$, each vertex of K_i^0 will also do so, since they have at least $3d - 1$ inactive neighbors, and at most $2d + 1$ active ones. Now, every vertex of K_1^0 has $3d - 1$ inactive and $3d - 1$ active neighbors, plus the special vertex $v = v_1$. If we have $x(t)_v = 1$ for some $t > 0$, then necessarily $x(t')_w = 1$ for every vertex $w \in K_0^0$ and every $t' \geq t$. At time t, every vertex of K_2^0 then has at least $3d$ active and at most $2d$ inactive neighbors, so they

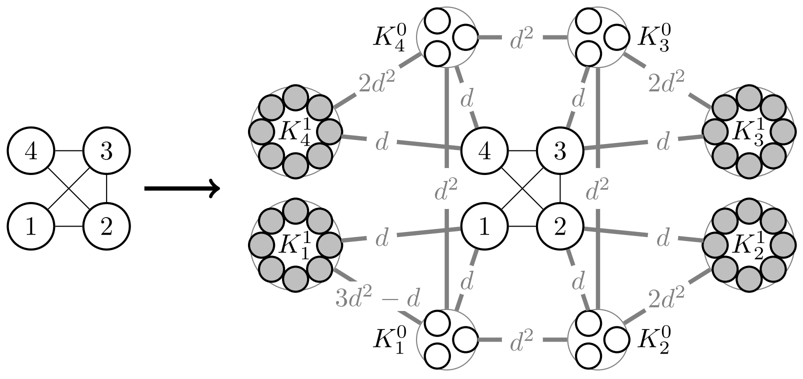

Fig. 8. The construction of Theorem 13 applied to a simple example graph of degree $d = 3$. The circular formations denote complete subgraphs, and the thick gray lines denote multiple edges (their number is given in the label).

will become active on the next time step, and stay that way. Inductively, every vertex of $H \setminus G$ will become active, and after that, so will the vertices of G, since d is an upper bound for their degree in G. Thus the graph H will eventually contain only active vertices if and only if $x(t)_v = 1$ for some time $t > 0$. □

5. Moving the threshold

In this section we will show that our results are valid for more general update rules than the majority rule. For a real number $p \in (0,1)$ and a graph G, consider the rule obtained from the following function for each vertex $v \in V(G)$:

$$f_i^p(x) = \begin{cases} 1, & \text{if } \sum_{j\in N(i)} x_j > p|N(i)|, \\ 0, & \text{if } \sum_{j\in N(i)} x_j \le p|N(i)|, \end{cases}$$

where $N(v) \subset V(G)$ is the set of neighbors of v in G. The rule obtained from this vertex function is called *portion-p* rule, and the automata network with the portion-p rule will be called a *portion-p network*. Notice that the majority network is a portion-1/2 network. The dynamics of a portion-p network is denoted by F_p. For this class of functions, we can define the one cell prediction problem for a fixed $p \in (0,1)$ and a family $\mathcal{S}$ of updating schemes:

$\mathcal{S}$-Ocp(p): Given a portion-p network $\mathcal{A} = (G, (f_v^p)_{v\in V})$, an updating scheme $S \in \mathcal{S}$, a configuration $x \in \{0,1\}^V$ of $\mathcal{A}$, and an initially inactive vertex $v \in V(G)$, does there exist $y \in T_S(x)$ such that $y_v = 1$?

Using the same arguments given for the majority network, for any portion $p \in (0,1)$ the problems Seq-Ocp(p) and Syn-Ocp(p) can be solved in polynomial time by simply simulating the dynamics of the network. In the following, we will show that for any $p \in (0,1)$ satisfying the appropriate computability conditions, the block sequential version Bseq-Ocp(p) is **PSPACE**-complete.

Theorem 14. *For all $p \in (0,1)$ whose digits can be computed in polynomial space, the problem* Bseq-Ocp(p) *is* ***PSPACE*****-complete.**

Proof. Let $p \in (0,1)$, and assume $p < 1/2$, the other case being essentially symmetric. We will reduce Bseq-Majority to Bseq-Ocp(p). Let $\mathcal{A}$ be a majority network, G the underlying graph of $\mathcal{A}$, S_G a block sequential updating scheme, $x \in \{0,1\}^V$ a configuration of $\mathcal{A}$ and $v \in V(G)$.

For each vertex $w \in G(V)$, denote $d_G(w) = |N_G(w)|$, and let $n(w)$ be any integer satisfying

$$\lfloor p(d_G(w) + n(w)) \rfloor = \left\lfloor \frac{d_G(w)}{2} \right\rfloor, \tag{1}$$

which exists since $p < 1/2$. We also note that $pn(w) < (1/2 - p)d_G(w) + 1$ holds, so that $n(w)$ is at most linear in $d_G(w)$ when p is a constant. Now, we construct a new graph H from G by attaching to it a complete graph K_w of $\max(n(w), \lceil 1/p \rceil + 1)$ inactive vertices for each $w \in V(G)$, and adding a new edge from w to $n(w)$ vertices of K_w.

We now claim that the subgraph G of the portion-p network $(H, (f_w)_{w\in V(H)})$ behaves identically to the majority network $\mathcal{A}$, if the new vertices are inactive, which then proves the claim. For this, it suffices to note that for all such configurations $y \in \{0,1\}^{V(H)}$ and all $w \in V(G)$ we have $\sum_{u\in N_H(w)} y_u = \sum_{u\in N_G(w)} y_u$, and by (1), these integers are larger than $\frac{1}{2}d_G(w)$ if and only if they are larger than $pd_H(w)$. Thus we have $F_p(y)_w = F(y|_G)_w$, and by induction $F_p^t(y)_w = F^t(y|_G)_w$

for all $t \geq 0$. In particular, if we denote by $x' \in \{0,1\}^{V(H)}$ the extension of x to H, then $F^t(x)_v = 1$ holds if and only if $F^t_p(x')_v = 1$ does. $\square$

We note that in the above reduction, it is essential that ties are handled the same way in majority networks and portion-p networks.

6. Conclusion

In this article, we have studied the computational complexity of predicting the evolution of Boolean majority networks under the block sequential update schemes. It turns out that, while the sequential and synchronous schemes admit a polynomial-time prediction algorithm, predicting the behavior of a single vertex is **PSPACE**-complete in the more general case of block sequential schemes. This is due to two factors: first, arbitrary Boolean circuits can be simulated by majority networks (which is also why the prediction of the sequential and synchronous schemes is **P**-complete [7]), and second, the block sequential scheme allows us to construct small gadgets, which we call clocks, that empty the circuit once its computation is finished, and transport the output of the computation back into the input vertices, so that the circuit can be iterated indefinitely. We also argue that the one cell prediction problem is a fundamental one by reducing other prediction problems to it, namely, the prediction of one cell being eventually active for some time on, the prediction of all cells becoming active, and the one cell prediction problem for networks with different thresholds.

We have tried to minimize the maximum degree, block size, and the number of blocks in our theorems to some extent, but it is very likely that they can be further improved. We leave it as an open problem to find the minimum values for there parameters that still keep the problems **PSPACE**-complete.

A natural continuation of this research would be to study Boolean networks with even more general update rules. For example, consider the case of *AND-OR networks*, which are Boolean automata networks $\mathcal{A} = (G, (f_v)_{v \in V})$ where each vertex function f_v computes either $\min N_G(v)$ or $\max N_G(v)$. In such networks, implementing a monotone Boolean circuit is already difficult, since a single inactive neighbor, causes a min-vertex to become inactive in the next step, but every simulated gate should somehow be connected to the vertices that correspond to its outputs. This problem has been overcome in [7], where it was shown that for sequential and synchronous schemes, the prediction problem of AND-OR networks is in fact **P**-complete. With block sequential schemes, there again exist limit cycles of exponential lengths, which suggests that the prediction problem may be **PSPACE**-complete in this case.

References

[1] Sanjeev Arora, Boaz Barak, Computational Complexity: A Modern Approach, 1st edition, Cambridge University Press, New York, NY, USA, 2009.
[2] Stefan Bornholdt, Boolean network models of cellular regulation: prospects and limitations, J. R. Soc. Interface 5 (Suppl 1) (2008) S85–S94.
[3] Claudio Castellano, Santo Fortunato, Vittorio Loreto, Statistical physics of social dynamics, Rev. Modern Phys. 81 (May 2009) 591–646.
[4] Maria I. Davidich, Stefan Bornholdt, Boolean network model predicts cell cycle sequence of fission yeast, PLoS ONE 3 (2) (2008) 02.
[5] Eric Goles, Pedro Montealegre, Computational complexity of threshold automata networks under different updating schemes, Theoret. Comput. Sci. 559 (0) (2014) 3–19.
[6] E. Goles-Chacc, Comportement oscillatoire d'une famille d'automates cellulaires non uniformes, Université scientifique et médicale de Grenoble, Institut national polytechnique de Grenoble, 1980.
[7] E. Goles-Chacc, F. Fogelman-Soulie, D. Pellegrin, Decreasing energy functions as a tool for studying threshold networks, Discrete Appl. Math. 12 (3) (1985) 261–277.
[8] Nicolás Goles Domic, Eric Goles, Sergio Rica, Dynamics and complexity of the Schelling segregation model, Phys. Rev. E 83 (May 2011) 056111.
[9] R. Greenlaw, H.J. Hoover, W.L. Ruzzo, Limits to Parallel Computation: P-Completeness Theory, Oxford University Press, 1995.
[10] J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, Proc. Natl. Acad. Sci. USA 79 (8) (1982) 2554–2558.
[11] Henning S. Mortveit, Christian M. Reidys, An Introduction to Sequential Dynamical Systems, Universitext, Springer, 2008.