A Fast Parallel Algorithm for the Robust Prediction of the Two-Dimensional Strict Majority Automaton

Eric Goles[1,3(✉)] and Pedro Montealegre[2]

[1] Facultad de Ingeniería Y Ciencias, Universidad Adolfo Ibáñez, Santiago, Chile
eric.chacc@uai.cl
[2] Université D'Orléans, INSA Centre Val de Loire, LIFO EA, 4022 Orléans, France
[3] Le Studium: Loire Valley Institute for Advanced Studies, Orléans, France

Abstract. Consider the *robust prediction* problem for some automaton as the one consisting in determine, given an initial configuration, if there exists a nonzero probability that some selected site change states, when the network is updated picking one site at a time uniformly at random. We show that the *robust prediction* is in **NC** for the two-dimensional, von Neumann neighborhood, *strict majority automaton.*

Keywords: Majority automata · Prediction problem · Asynchronous automata · Computational complexity · Fast parallel algorithm · Bootstap percolation

1 Introduction

The study of the dynamics of cellular automata and their relation with the computational complexity was introduced, to our knowledge, by Banks in the 70's [1,2]. The computational complexity of a cellular automaton is defined as its capability to simulate algorithms using certain configurations. In other words, to be Turing-Universal. These notions can be translated into decision problems, consisting in the prediction of state changes in some site, given an initial configuration. The complexity of the automaton is then related with the complexity of this type of decision problems, where the Turing Universality is translated as the **P-Completeness** of the prediction problem.

The class **NC**, which is a subclass of **P**, is known as the class of problems that can be solved with a *fast parallel algorithm*, which run in poly-logarithmic time in a PRAM machine using a polynomial number of processors. It is widely believed that $\mathbf{NC} \neq \mathbf{P}$. The membership of the prediction problem of some **CA** in **NC**, suggest that this automaton is not capable of simulating algorithms in the sense above, since it can only simulate very simple circuits.

These perspectives have been studied by several authors, in the context of *sand piles* and the *chip firing game* [3–5,14], the *majority automaton* [13] and the *life without death* [11] in cellular automata, as well as when the dynamics

S. El Yacoubi et al. (Eds.): ACRI 2016, LNCS 9863, pp. 166–175, 2016.
DOI: 10.1007/978-3-319-44365-2_16

are not necessarily defined over a finite lattice but over some graph [7,9]. These results usually consider only synchronous dynamics, i.e., at each step the sites are updated at the same time. However, there exist some results around prediction problems with different updating schemes [6,8].

Another prediction problem considers the so-called *fully asynchronous* updating schemes, where in each time step a single site is updated, picked uniformly at random. The prediction problem in this context (that we call *robust-prediction*) consists in determining, given an initial configuration, if there exists a nonzero probability that some site change states. In other words, determine if there exists a sequence of site updates that produces the selected node to change states. In [13] it is suggested that this prediction problem belongs to the class **NC** restricted to two-dimensional majority cellular automata.

In this paper we show that the *robust prediction* problem is effectively in **NC** for the two-dimensional *strict majority automaton* with the von Neumann neighborhood. Our result is based on an algorithm on [9], which is used to solve a version of the prediction problem (the synchronous one) in the *freezing* version of the two-dimensional strict majority automaton. *Freezing* means that all sites that begin in or reach state 1, remain in that state forever.

In next section we begin by giving some formal definitions of the concept exposed above. In Sect. 3 we show the main result, and in Sect. 4 we give some conclusions.

2 Preliminaries

In the following $[n]$ denotes the set $\{1,\dots,n\}$. For a node v in a graph $G = (V, E)$, we call $N(v)$ the neighborhood of v and $N[v] = N(v) \cup \{v\}$ the closed neighborhood of v. Let $U \subseteq V$ be a set of nodes, then $G[U]$ is the subgraph of G induced by the nodes in U.

An automata network (**AN**) of size n is a tuple $A = (G, F)$, where $G = (V, E)$ is a graph of size n. Each node $v \in V$ has a *state* in $\{0, 1\}$. Nodes in state 1 are called *active* while nodes in state 0 are called *inactive*. A *configuration* x of G is a state assignment to each node of G, represented by an element of $\{0, 1\}^n$. The configurations evolve according to a *global function* $F : \{0, 1\}^n \to \{0, 1\}^n$, composed of *node functions* $(f_v)_{v\in V(G)}$ such that $f_v : \{0, 1\}^{N[v]} \to \{0, 1\}$ depends only in the closed neighborhood of a node and $F(x) = (f_v(x))_{v\in V}$. Usually F is called the *rule* of the automata network. When G is a finite two-dimensional lattice with periodic boundary conditions, we say that the corresponding **AN** is a two-dimensional cellular automaton (**CA**). In this paper we will always consider the von Neumann neighborhood for **CA**'s (i.e., the four sites orthogonally surrounding a central site).

An *updating scheme* of an **AN** of size n is a function $\sigma : \mathbb{N} \to 2^{[n]}$, which defines which nodes are updated at each time step. An updating scheme σ is applied to an automata network $A = (G, F)$ as follows: define for $u \in V(G)$ and

$i \in \mathbb{N}$ the value of $(F^{\sigma(i)}(x))$ in the site u as:

$$(F^{\sigma(i)}(x))_u = \begin{cases} (F(x))_u & \text{if } u \in \sigma(i), \\ x_u & \text{otherwise.} \end{cases}$$

and $F^{\sigma,i}(x) = F^{\sigma(i)}(F^{\sigma,i-1}(x))$ with $F^{\sigma,0}(x) = x$. The synchronous updating scheme, represented by the function *sync* $\equiv [n]$, corresponds to the one where each site is updated at each time step. We denote $F^{sync,i} = F^i$, for any $i > 0$. In this paper all updating schemes which are not the synchronous one will be *sequential*, this is, for any $t > 0$ $|\sigma(t)| = 1$. In words, this means that at each time step we will update a single site. Notice that we do not ask the sequential updating schemes to be *fair*: some nodes may be updated several times, while others may not be updated at all. A sequential updating scheme is called *fully asynchronous* if at each time step the updated node is picked uniformly at random.

A node $v \in V$ is called *stable* for a pair (G, x) and some updating scheme σ of rule F, where G is a graph and x is a configuration of G, if $(F^{\sigma(t)}(x))_v = x_v$ for all $t > 0$. For any two n dimensional boolean vectors, $x, y \in \{0,1\}^n$, we say that $x \leq y$ if $x_i \leq y_i$ for each $i \in [n]$. A rule F is called *monotone* if for each $x^1, x^2 \in \{0,1\}^n$, $x^1 \leq x^2$ implies $F(x^1) \leq F(x^2)$. For any local rule F, we define the *freezing* version of F, denoted $\overline{F}$, as the rule defined as $(\overline{F}(x))_v = 1$ if $x_v = 1$ and $(F(x))_v$ otherwise.

The *Strict Majority* rule is the global function defined using the following local functions:

$$f_v(x) = \begin{cases} 1 & \text{if } \sum_{u \in N(v)} x_v > |N(v)|/2, \\ 0 & \text{otherwise.} \end{cases}$$

We call *Bootstrap Percolation* the freezing version of the strict majority rule.

2.1 Prediction Problems

As we said in the introduction, the computational complexity of an automata network can be characterized by different prediction problems. Let G be a graph and $v \in V(G)$ a special node, that we will call the *objective node*. In the following we will define three prediction problems, each of them consisting in deciding if a objective node will change states, given an initial configuration that evolves according to some fixed rule.

The first problem, called PREDICTION consists in determining if the objective node will change states in at most some given number of synchronous steps, given also an initial configuration. Formally the problem is stated as follows:

PREDICTION(F)
Input: A graph G, an initial configuration x, a vertex $v \in V(G)$ such that $x_v = 0$ and $T > 0$.
Question: Determine if $(F^T(x))_v = 1$.

The second problem, called EVENTUAL-PREDICTION is similar to PREDICTION. In this problem, the number of steps to decide a state change is not given, so the problem consists in determining if there exists a time step in which the objective node becomes active. In other words we ask if the objective node is stable for the synchronous updating. Formally the problem is stated as follows:

EVENTUAL-PREDICTION(F)
Input: A graph G, an initial configuration x and a vertex $v \in V(G)$ such that $x_v = 0$.
Question: Does there exists $T > 0$ such that $(F^T(x))_v = 1$?

The third problem is called ROBUST-PREDICTION, and the question is if there exists a nonzero probability that the objective node becomes active in at most some given number of steps, when the network is updated in a fully asynchronous updating scheme. In other words, if there exists a sequential updating scheme that makes the objective node to change. Formally the problem is stated as follows:

ROBUST-PREDICTION(F)
Input: A graph G, an initial configuration x, a vertex $v \in V(G)$ such that $x_v = 0$, and $T > 0$.
Question: Does there exists a sequential updating scheme σ such that $(F^{\sigma,T}(x))_v = 1$?

Let $\mathcal{G}$ be a family of graphs. We call PREDICTION (respectively EVENTUAL-PREDICTION, ROBUST-PREDICTION) restricted to $\mathcal{G}$ to the decision problem PREDICTION (respectively EVENTUAL-PREDICTION, ROBUST-PREDICTION) where the input graph is restricted to belong to $\mathcal{G}$.

Notice that PREDICTION(F) belongs to the class **P** for any rule F (whose node function is computable in polynomial time), because simulating the automaton for the given number of steps we can obtain the answer. Several examples exist for rules that are **P-Complete** both in general graphs and restricted to the two-dimensional case [9,11]. On the other hand, EVENTUAL-PREDICTION(F) is in general in **PSPACE**, and may be **PSPACE-Complete**. However, when the local function is a threshold function with symmetric weights (for example the strict majority rule), the problem becomes polynomial, since in that case the automaton reaches in at most a polynomial number of synchronous steps, a fixed point or a two-cycle [10]. Finally ROBUST-PREDICTION(F) is in general in **NP**, since a sequential updating scheme that makes the objective node active is a certificate that can verified in polynomial time.

In the following we will show that for the Strict Majority and Bootstrap Percolation rules, some of these problems restricted to the family of regular graphs of degree four are in **NC**.

2.2 Parallel Subroutines

In our algorithms we will use as subroutines some fast parallel algorithms to compute graph properties. These algorithms can be found in [12]. The connected components of a graph G are the equivalence classes of the connectivity relation over the vertices of G. The biconnected components of a graph are the equivalence classes of the relation over the edges of G, where two edges are related if they are both contained in the same cycle.

Proposition 1 ([12]). *There exist the following fast parallel algorithms:*

- **A Connected-Components algorithm**, *that receives as input the adjacency matrix of a graph of size n, and returns an array C of dimension n, such that $C(i) = C(j)$ if and only if nodes v_i and v_j are in the same connected component. The algorithm runs in time $\mathcal{O}(\log n)$ using $\mathcal{O}(n^2 \log n)$ processors in a CRCW PRAM*[1]*.*
- **A Biconnected-Components algorithm**, *that receives as input the adjacency matrix of a connected graph of size n, and returns an array B of dimension $\binom{n}{2}$, such that $B(e_i) = B(e_j)$ if and only if edges e_i and e_j are in the same biconnected component. The algorithm runs in time $\mathcal{O}(\log^2 n)$ using $\mathcal{O}(n^2)$ processors CRCW PRAM.*
- **A Rooting-tree algorithm**, *that receives as input a tree T represented as the adjacency lists of its vertices and a special node r of T, and returns for each node v the value $p(v)$ which corresponds to the parent of v in the tree T rooted at r. The algorithm runs in time $\mathcal{O}(\log n)$ using $\mathcal{O}(n)$ processors in a CRCW PRAM.*
- **A Tree-level algorithm**, *that receives as input a tree T represented as the adjacency lists of its vertices, a special node r of T, and returns for each node v the value $l(v)$ which corresponds distance of node v to root r. The algorithm runs in time $\mathcal{O}(\log n)$ using $\mathcal{O}(n)$ processors in a CRCW PRAM.*

We will also use a fast parallel algorithm to solve Eventual-Prediction for the Bootstrap Percolation rule.

Proposition 2 ([9]). *Let $\overline{F}$ be the Bootstrap Percolation rule. There exists an algorithm that solves* Eventual-Prediction$(\overline{F})$ *restricted to regular graphs of degree four, in time $\mathcal{O}(\log^2 n)$ using $\mathcal{O}(n^4)$ processors in a CRCW PRAM.*

This fast parallel algorithm, that we call **Algorithm 1** is based in the following characterization of nodes that are stable for the synchronous update of the Bootstrap Percolation rule.

Proposition 3 ([9]). *Let G be a regular graph of degree four, x be a configuration of G, and $v \in V(G)$ some node such that $x_v = 0$. Call $G[0, v]$ the connected component of $G[\{u \in V(G) : x_u = 0\}]$ that contains v. Then v is stable for (G, x) updated synchronously with the Bootstrap Percolation rule if and only if either v belongs to a cycle or to a path between two cycles in $G[0, v]$.*

[1] Concurrent-read Concurrent-write Parallel Random Access Machine: A RAM with several processors, which can read and write a shared memory.

Then, the fast parallel algorithm of [9] consists in finding the connected and biconnected components of the graph induced by the nodes initially inactive, and then detect if the objective node belongs to some cycle or a path between two cycles in that graph. If it is not, then from the above proposition necessarily the node eventually change states. In Sect. 3 we show that we can adapt this algorithm to find, in case that the objective node is not stable, the exact number of nodes that we must update in order to produce a change in the state of the objective node.

3 The Strict Majority Rule

Before enunciate the main theorem, we prove the following lemma.

Lemma 1. *Let G be a graph, x be a configuration of G, and $v \in V(G)$ a vertex such that $x_v = 0$. Then v is a stable node for the synchronous updating scheme of the Bootstrap Percolation rule if and only if v is stable for any sequential updating scheme of the Strict Majority rule.*

Proof. We will show that the property is true for any monotone rule F and its freezing version $\overline{F}$. We notice first that the monotonicity of F implies that $F^{\sigma,i}(x) \leq \overline{F}^i(x)$, for any $i > 0$, and sequential updating scheme σ. Indeed, from the definition of $\overline{F}$, for any configuration $y \in \{0,1\}^n$ and $k \in \mathbb{N}$, $F^{\sigma(k)}(y) \leq \overline{F}(y)$, in particular $F^{\sigma(1)}(x) \leq F(x)$. If we suppose then $F^{\sigma,i-1}(x) \leq \overline{F}^{i-1}(x)$, we obtain

$$F^{\sigma,i}(x) = F^{\sigma(i)}(F^{\sigma,i-1}(x)) \leq F^{\sigma(i)}(\overline{F}^{i-1}(x)) \leq \overline{F}^i(x).$$

where the first inequality follows from the monotonicity of F, and the second one from the base case. We obtain that if there exists an updating scheme σ and a time $t > 0$ such that $(F^{s,t}(x))_v = 1$, then $(\overline{F}^t(x))_v = 1$.

In the other direction, suppose that there exists a time $t > 0$ such that $(\overline{F}^t(x))_v = 1$. Call $U_i = \{u \in V(G) : (\overline{F}^{i-1}(x))_u = 0 \wedge (\overline{F}^i(x))_u = 1\}$ the set of the nodes that change states in step i (from 0 to 1, since $\overline{F}$ is *freezing*). Call now σ the updating scheme where sequentially update one by one the nodes in $U_1, \ldots, U_t$, starting from U_1, and choosing for each $i \in \{1, \ldots, t\}$ an arbitrary order in U_i. Call $t^* = \sum_{i=1}^t |U_i|$, then we have that for each $u \in U_i$ and $\sum_{j=1}^i |U_j| \leq k \leq t^*$, $(F^{\sigma,k}(x))_u = (\overline{F}^i(x))_u = 1$. Then, in particular $(F^{\sigma,t^*}(x))_v = 1$. $\square$

Theorem 1. *Let F be the Strict Majority rule. Then* Robust-Prediction(F) *restricted to the family of regular graphs of degree four is in* ***NC***.

Proof. Let (G, x, v, T) be an input of the Robust-Prediction(F) problem. We start checking, using **Algorithm 1**, that v is not stable in G and x for the Bootstrap Percolation rule updated with a synchronous updating scheme. If it does, our algorithm returns, indicating that the objective node is stable for any updating scheme of the strict majority rule.

In the following we suppose that v is not stable for the synchronous updating of Bootstrap Percolation rule. From Lemma 1 this means that there exists a sequential updating scheme σ for which v is not stable for the strict majority rule updated according to σ. This scheme will consist in the sequential updating of the nodes that change states at each step of the synchronous updating of the bootstrap percolation rule. We choose an arbitrary ordering if several nodes are updated at the same time step.

Let v_1, v_2, v_3 and v_4 the neighbors of v in G. Let $G[0] = G[\{u \in V(G) : x_u = 0\}]$ the induced graph of nodes initially inactive, and call $G[0; v]$ the connected component of $G[0]$ containing v. For $i \in \{1, 2, 3, 4\}$ call T_i the connected component of $G[0; v] - v$ that contain v_i.

Without loss of generality, suppose that v_1, v_2 and v_3 are the first three neighbors of v to become active before v in the synchronous update of the bootstrap percolation rule (in particular they can be initially active, in that case $T_i = \emptyset$). Notice that if $T_i \neq \emptyset$ then T_i is a tree. Indeed, if T_i contain a cycle C_i, then either v_i is contained in C or v_i is in a path P between C and v. In both cases each internal node of the path P will have two inactive neighbors, so v_i cannot change before v in a synchronous update of the Bootstrap Percolation rule, a contradiction.

Call T_i^0 the tree T_i, and for $t > 0$ call T_i^t the subtree of T_i that contains the inactive nodes after t synchronous updates of the bootstrap percolation rule. Notice that for any $t > 0$, the set $V(T_i^{t-1}) \backslash V(T_i^t)$ is the set of leafs of T_i^{t-1}, except possibly v_i. Indeed in time $t-1$ each internal node of T_i^{t-1} has at least two inactive neighbors, every leaf has three active neighbors, and v_i is adjacent to v, which we suppose to become active after v_i. This implies that v_i will become active only once each node of T_i becomes active. Notice that if we sequentially update the leafs of the trees $T_i^0, T_2^1, \dots$ we obtain a sequential updating scheme which produces a change of states in v_i in the minimum number of steps. Indeed, if there is a faster sequential updating scheme, that means that there is some $t > 0$ such that one leaf T_i^t was not updated, so it remains inactive. That implies that internal nodes of a path between such non updated leaf and v_i will have at least two inactive neighbors, preventing v_i to become active. We conclude that the minimum number of sequential steps to produce v_i to change states is $|T_i|$. We obtain that the objective node v changes in at most T steps if and only if $|T_1| + |T_2| + |T_3| + 1 \leq T$ (Fig. 1).

Let G be a regular graph of degree 4, v a node of G, and let v_1, v_2, v_3 and v_4 be the neighbors of v. We call G^{v,C_4} the graph obtained from G removing v, and replacing it with a cycle of length 4, called $C_4 = \{c_1, c_2, c_3, c_4\}$, such that for each $i \in \{1, 2, 3, 4\}$, the edge $\{v, v_i\}$ is replaced in G^{v,C_4} with $\{c_i, v_i\}$. Let x be a configuration of G such that $x_v = 0$, we call x^{v,C_4} the configuration of G^{v,C_4} such that $x_w^{v,C_4} = x_w$ if $w \neq v$ and $x_{c_i}^{v,C_4} = 0$ for $i \in \{1, 2, 3, 4\}$. Notice that from Proposition 3, T_i contains a cycle if and only if v_i is stable for (G^{v,C_4}, x^{v,C_4}) and the synchronous updating of the Bootstrap Percolation rule. We are now ready to present **Algorithm 2**.

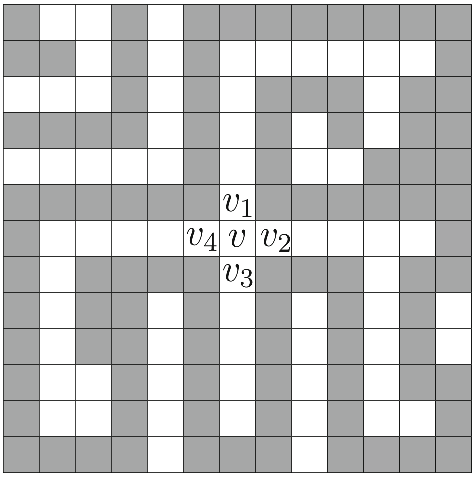

Fig. 1. Example of a configuration in the two-dimensional lattice. The objective node is v, which is not stable for the synchronous updating of the bootstrap percolation rule. Note that $|T_1| = 12$, $|T_2| = 11$ and $|T_3| = 5$, and that T_4 contains a cycle.

Algorithm 2

Input: A graph G of size n represented by its adjacency list, an array of dimension n representing a configuration x, a vertex $v \in V(G)$ and $T > 0$ such that $x_v = 0$.

Output: A boolean *out* that indicates if there exists an updating scheme σ such that $(F^{\sigma,T}(x))_v = 1$, where F is the Strict Majority rule.

```
1 Define for $i = \{1,2,3,4\}$, $k_i \leftarrow \infty$.
2 Run Algorithm 1 to check $v$ is stable for $(G,x)$ and the synchronous update of the bootstrap percolation rule. If it does not then continue, else return out ← false.
3 Build the adjacency matrix of $G^{v,C_4}$ and the configuration $x_{v,C_4}$.
4 For each $v_i$ neighbor of $v$ do in parallel:
  4.1 If $x_{v_i} = 1$ then $k_i \leftarrow 0$ and exit the for loop else contiue.
  4.2 Run Algorithm 1 on input $(G^{v,C_4}, x^{v,C_4}, v_i)$ to check if $v_i$ is stable for $(G^{v,C_4}, x^{v,C_4})$ and the synchronous update of the bootstrap percolation rule.
  4.3 if $v_i$ is not stable then
    4.3.1 Use the Connected Components Algorithm of Proposition 1 to compute $T_i$, the connected component of $G[0,v]-v$ containing $v_i$.
    4.3.2 $k_i \leftarrow |T_i|$
5 Compute $t_1 \leftarrow \min\{k_1,k_2,k_3,k_4\}$, $t_2 \leftarrow \min\{k_1,k_2,k_3,k_4\}\backslash\{t_1\}$,
  $t_3 \leftarrow \min\{k_1,k_2,k_3,k_4\}\backslash\{t_1,t_2\}$ and $T^* \leftarrow t_1+t_2+t_3+1$.
7 If $T^* < T$ then out ← true else out ← false
```

The correctness of **Algorithm 2** is follow from the paragraphs above. About the complexity, **Algorithm 2** runs $\mathcal{O}(\log^2 n)$ time with $\mathcal{O}(n^4)$ processors, where the most expensive part is the runnings of **Algorithm 1** in steps **2** and **4.2**, running in $\mathcal{O}(\log^2 n)$ time with $\mathcal{O}(n^4)$ each. Steps **1**, **3**, **5** and **7** can be done in $\mathcal{O}(\log n)$ sequential time, steps **4.1**, **4.3.2** can be done in $\mathcal{O}(\log n)$ time using 4 processors. Finally step **4.3.1** can be done in time $\mathcal{O}(\log^2 n)$ and $\mathcal{O}(n^2)$ processors according to Proposition 1. □

Remark 1. Notice that we can easily adapt **Algorithm 2** to output, in case that the objective node is not stable, the updating scheme that makes it change in the wished number of steps. Indeed, after step **4.3.2** we can compute for each subtree T_i the level of its nodes with respect to the root v_i, using the algorithms cited in Proposition 1. The updating scheme is then defined in decreasing order with respect to the level of the nodes, where tie cases are solved arbitrarily.

Corollary 1. *Let F be the strict majority rule. Then* Robust-Prediction(F) *restricted to the two-dimensional lattice is in NC.*

Remark 2. We can also adapt **Algorithm 2** to solve Prediction for the Bootstrap Percolation rule, restricted to the family of regular graphs of degree 4. Again, we can compute after step **4.2** the level of each node with respect to the corresponding subtree. The algorithm outputs *true* if there are three non stable neighbors of v whose trees have depth smaller than T.

Corollary 2. *Let F be the bootstrap percolation rule. Then* Prediction(F) *restricted to the two-dimensional lattice is in NC.*

4 Discussion

We had proven that the Robust-Prediction is in **NC** for the strict majority rule in the two dimensional lattice. This suggests, for example, that unless **NC** = **P** a two dimensional strict majority **CA** can not simulate monotone circuits that are not planar. However, the complexity of Prediction for the strict majority rule restricted to a two dimensional lattice is open. In [13] is conjectured that this problem is not **P-Complete**. The possibility that a monotone function, in two dimensions, and with a von Neumann neighborhood is capable of simulating non planar circuits is unlikely.

What about higher dimensions? In [9] it is shown that Prediction is **P-Complete** for the Bootstrap percolation rule for the family of graphs that admit nodes of degree greater than 4, and in [13] it is shown that in Prediction is **P-Complete** for the majority rule in the d-dimensional lattice, for $d \geq 3$. In the case of Robust-Prediction, the problem is in general in **NP**. In a future work we will show that when we are not restricted to a two dimensional lattice, both the Bootstrap Percolation and the Strict Majority rules are **NP-Complete**.

References

1. Banks, E.R.: Universality in cellular automata. In: SWAT (FOCS). IEEE Computer Society, pp. 194–215 (1970)
2. Banks, E.R.: Information processing and transmission in cellular automata, Technical Report AITR-233, MIT Artificial Intelligence Laboratory (1971)
3. Gajardo, A., Goles, E.: Crossing information in two-dimensional sandpiles. Theor. Comput. Sci. **369**, 463–469 (2006)
4. Goles, E., Margenstern, M.: Sand pile as a universal computer. Int. J. Mod. Phys. C **07**, 113–122 (1996)
5. Goles, E., Margenstern, M.: Universality of the chip-firing game. Theor. Comput. Sci. **172**, 121–134 (1997)
6. Goles, E., Montealegre, P.: Computational complexity of threshold automata networks under different updating schemes. Theor. Comput. Sci. **559**, 3–19 (2014). Non-uniform Cellular Automata
7. Goles, E., Montealegre, P.: The complexity of the majority rule on planar graphs. Adv. Appl. Math. **64**, 111–123 (2015)
8. Goles, E., Montealegre, P., Salo, V., Törmä, I.: Pspace-completeness of majority automata networks. Theor. Comput. Sci. **609**(Part 1), 118–128 (2016)
9. Goles, E., Montealegre-Barba, P., Todinca, I.: The complexity of the bootstrapping percolation, other problems. Theor. Comput. Sci. **504**, 73–82 (2013). Discrete Mathematical Structures: From Dynamics to Complexity
10. Goles-Chacc, E., Fogelman-Soulie, F., Pellegrin, D.: Decreasing energy functions as a tool for studying threshold networks. Discrete Appl. Math. **12**, 261–277 (1985)
11. Moore, C.: Life without death is P-complete. Complex Syst. **10**, 437–447 (1996)
12. Jaja, J.: An introduction to parallel algorithms. Addison-Wesley Professional, New York (1992)
13. Moore, C.: Majority-vote cellular automata, Ising dynamics, and p-completeness. J. Stat. Phys. **88**, 795–805 (1997)
14. Moore, C., Nilsson, M.: The computational complexity of sandpiles. J. Stat. Phys. **96**, 205–224 (1999)