Theoretical Computer Science 504 (2013) 73–82

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

The complexity of the bootstraping percolation and other problems

Eric Goles[a], Pedro Montealegre-Barba[b,*], Ioan Todinca[c]

[a] *Facultad de Ciencias y Tecnología, Universidad Adolfo Ibáñez, Santiago, Chile*
[b] *Departamento de Ingeniería Matemática, Universidad de Chile, Correo 3, Santiago 170-3, Chile*
[c] *LIFO, Université d'Orléans, 45067 Orléans Cedex 2, France*

ARTICLE INFO

Keywords:
Computational complexity
Bootstrap percolation
Parallel algorithms
P-Completeness
Majority functions

ABSTRACT

We study the problem of predicting the state of a vertex in automata networks, where the state at each site is given by the majority function over its neighborhood. We show that for networks with maximum degree greater than 5 the problem is **P**-Complete, simulating a monotone Boolean circuit. Then, we show that the problem for networks with no vertex with degree greater than 4 is in **NC**, giving a fast parallel algorithm. Finally, we apply the result to the study of related problems.

1. Introduction

Let V a finite set of sites or vertices. An *Automata Network* defined on V is a triple $\mathcal{A} = (G, Q, (f_i : i \in V))$, where $G = (V, E)$ is a simple undirected graph, Q is the set of states, which is assumed to be finite, and $f_i : Q^{|V|} \to Q$ is the transition function associated to the vertex i. The set $Q^{|V|}$ is called the set of configurations, and the automaton's global transition function $F : Q^{|V|} \to Q^{|V|}$, is constructed with the local transition functions $(G, Q, (f_i : i \in V))$ and with some kind of updating rule, for instance a synchronous or a sequential one.

In the special case where $Q = \{0, 1\}$ we say that the state 1 means that the vertex is *active*, while state 0 represents *passive* vertices. If $N(v)$ is the neighborhood of v, i.e. the set of vertices $\{u \mid uv \in E\}$, consider the following transition function:

$$f_i(x) = \begin{cases} 1 & \text{if } x_i = 1 \\ 1 & \text{if } \sum_{j \in N(i)} x_j > \dfrac{|N(i)|}{2} \text{ and } x_i = 0 \\ 0 & \text{if } \sum_{j \in N(i)} x_j \le \dfrac{|N(i)|}{2} \text{ and } x_i = 0. \end{cases}$$

In other words, a passive vertex becomes active if the strict majority of its neighbors are active, and thereafter never changes its state. We call *the strict majority rule* to the global transition function given by $(f_i : i \in V)$.

This is a special case of bootstrap percolation, which corresponds to a simple model introduced in the late 1970s to study properties of some magnetic materials [1]. More recently it has been used to model problems like disease spreading [2], spreading of alerts on distributed networks [3], sand pile formation [4], and others [5].

The classical theoretical studies of bootstrap percolation deal with the question of what are the minimum number of active (infected) sites in a graph in order to infect, say with a high probability, the whole structure. The results in this

* Corresponding author. Tel.: +56 9 90991631.
E-mail addresses: eric.chacc@uai.cl (E. Goles), pedromb@gmail.com, pmontealegre@dim.uchile.cl (P. Montealegre-Barba), Ioan.Todinca@univ-orleans.fr (I. Todinca).

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.08.001

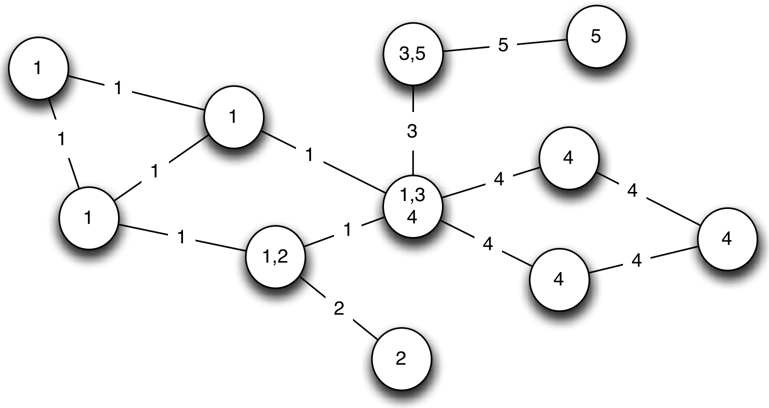

Fig. 1. Biconnected components of a graph. Edges with the same label are in the same component, notice that a vertex may appear in several biconnected components.

context are tricky and usually restricted to specific families of graphs (lattices, cubic graphs, etc) [6,7]. Our approach is different and complementary to previous one, since we ask for the possibility that an specific site will be infected given an initial configuration, and if that can be quickly predicted.

One of the first to introduce computation complexity to cellular automata (CA) and related systems was Moore [8–10]. This was done because, on one hand, it is very hard to obtain characterizations of the the general dynamical behavior of a CA on time; and on the other hand, they are related to parallel processing of information and algorithms, since some CA are able to emulate a universal Turing machine. Then, the idea is try to develop a bridge between these two aspects of the problem: its dynamical nature and its algorithmic capabilities.

The computational complexity of a prediction problem can be defined as the amount of resources, such as time or space, needed to predict it. In this case, we consider two fundamental classes: **P**, the class of problems solvable in polynomial time on a serial computer, and **NC**, the class of problems solvable in poly-logarithmic time in a PRAM machine, with a polynomial amount of processors. In other words, **NC** is the class of problems which have a fast parallel algorithm. It is a well known conjecture that **NC** $\neq$ **P** and, if so, there exist "inherently sequential" problems that belong to **P** and do not belong to **NC**. The most likely to be inherently sequential are **P**-Complete problems, to which any other problem in **P** can be reduced (by an **NC**-reduction or a logarithmic space reduction). If any of these problems has a fast parallel algorithm, then **P** = **NC** [11,8].

One such problem is the Circuit value problem (**CVP**), which consists in predicting the truth value of the output of a Boolean circuit, given the circuit and a truth value of its inputs. This problem is **P**-Complete since any deterministic Turing machine computation of length k can be converted into a Boolean circuit of depth k; thus polynomial time computations are equivalent to polynomial size and depth circuits; a complete analysis of this reduction can be found in [11]. The **CVP** remains **P**-Complete when the circuit is restricted to be monotone (that is, with AND and OR gates but without negation) and all vertices have in degree (fan in) and out degree (fan out) exactly two [11]. We call **M2CVP** this restriction of the **CVP** problem.

Now we give some definitions we need. Consider $G = (V, E)$ an undirected graph with vertex set V and edge set E. An edge between vertices x and y is simply denoted xy. We denote $n = |V|$ and $m = |E|$. Two vertices are *connected* if $u = v$ or there exists a path $P = x_1, x_2, \dots, x_k$ such that $u = x_1$, $v = x_k$ and $x_ix_{i+1} \in E$, for all $1 \leq i \leq k - 1$. This relation is an equivalence relation on V, and hence partitions V into equivalence classes $\{V_j\}_{j=1}^{l}$. The subgraphs $G_j = (V_j, E_j)$, where $E_j = \{xy \in E \mid x, y \in V_j\}$, are called the **connected components** of G. If G has a unique connected component, we say that G is connected.

Let $G = (V, E)$ be a connected graph. We define a relation R on E as follows. Given two edges e and g, eRg if and only if $e = g$ or e and g are in a common simple cycle. It is straightforward to check that R is an equivalence relation, and hence partitions E into equivalence classes $\{E_i\}_{i=1}^{s}$. Let $V_i = \{v \in V | v$ be the end point of some edge in $E_i\}$. Then, the subgraphs $G_i = (V_i, E_i)$ are called **biconnected components** of G (See Fig. 1).

It is important to remark that there are **NC** algorithms for finding connected and biconnected components of a given graph G [12].

Given $G = (V, E)$, and $v \in V$, the degree $d_G(v)$ of a vertex v is the number of its neighbors in G:

$$d_G(v) = |N(v)|$$

and the number $\Delta(G)$ is the maximum degree:

$$\Delta(G) = \max_{v \in V}\{d_G(v)\}.$$

As usual, the subscript will be omitted when no confusion is possible.

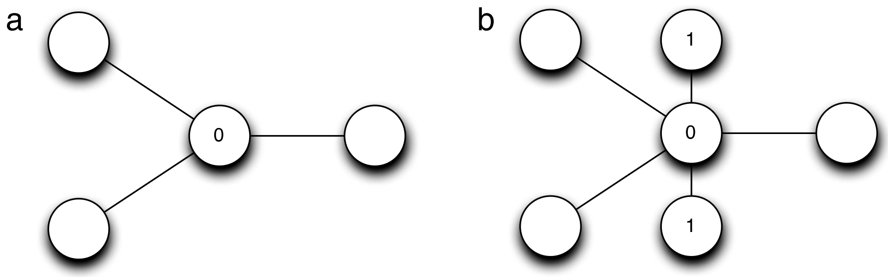

Fig. 2. (a) The AND gate. (b) The OR gate.

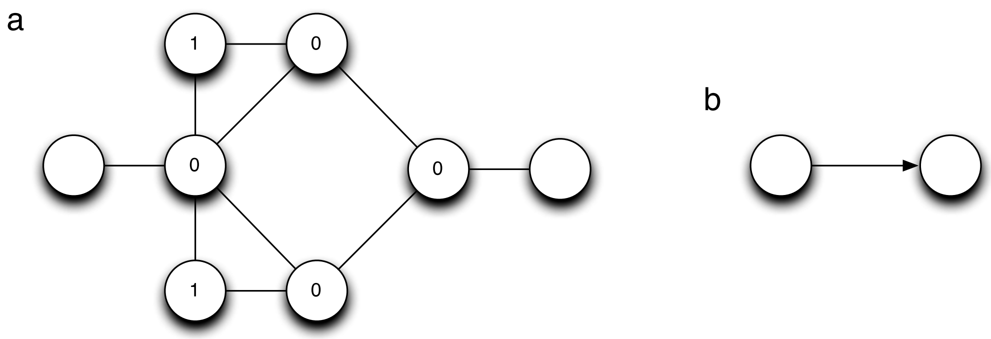

Fig. 3. (a) The diode. (b) Symbol of a diode.

In the next section, we discuss the complexity of the bootstrap percolation problem using the strict majority rule. In Section 4 we enlarge the results to other types of dynamics (simple majority, threshold rules). We end our paper with some open questions.

2. Complexity of bootstrapping percolation

Consider the following problem: given an initial configuration $x(0) \in \{0, 1\}^{|V|}$ and a vertex $v \in V$, initially passive ($x_v(0) = 0$), determine if there exists a time $T > 0$ such that v is active ($x_v(T) = 1$), where $x(t) = F(x(t-1))$ and F is some synchronous global transition function (for example, the strict majority rule). We call this decision problem **PER**.

Our main result is the following:

Theorem 1. *For the strict majority rule:*

1. *On the family of graphs $G = (V, E)$ such that $\Delta(G) \geq 5$, the problem* **PER** *is* ***P****-Complete.*
2. *On the family of graphs $G = (V, E)$ such that $\Delta(G) \leq 4$, the problem* **PER** *is in* ***NC****.*

We prove here the first part of the theorem, the second is postponed to the next section.

Proof of Theorem 1 (1). Notice that since the active vertices remains active, in at most $|V|$ steps, the dynamic gets into a fixed point. Then, **PER** can be decided in $\mathcal{O}(n)$, so is in **P**.

To prove that **PER** is **P**-Complete, we will reduce the restricted case circuit value problem **M2CVP** to **PER**. Since **M2CVP** is **P**-Complete, if the reduction uses only a logarithmic space, then **PER** will be **P**-Complete.

To reduce **M2CVP** to **PER**, we will build, given a monotone circuit, a graph that simulate its gates, where active vertices represent Boolean value *true*, and passive vertices represent value *false*. Since the circuit is monotone, we only have to simulate the AND and OR gates.

The AND gate (Fig. 2(a)) is simulated by an initially passive middle vertex with degree 3, two of them will be the inputs and the other the output. By the strict majority rule, this vertex will become active only if two more neighbors become active. In a similar way, the OR gate (Fig. 2(b)) has an initially passive middle vertex, with degree 5 and two neighbors initially active, then, this vertex will become active if any passive neighbor becomes active.

In order to avoid problems with the flow of information, Fig. 3(a) shows the construction of a *diode* which only allows the flow of information in ‘one way’: If the left vertex is active, then the right vertex will become active. But if the right vertex is active, the left vertex remains in its state. We simplify the notation with an arrow Fig. 3(b).

We modify the original gates, in order to have precise definitions of inputs and outputs. Fig. 4 shows the constructions of AND an OR gates with diodes.

Remember that in our restricted version of the circuit value problem **M2CVP**, every gate of the original circuit must have fan-out exactly 2, so we use OR gates ‘backwards’ in order to multiply the information of the output, as in Fig. 5.

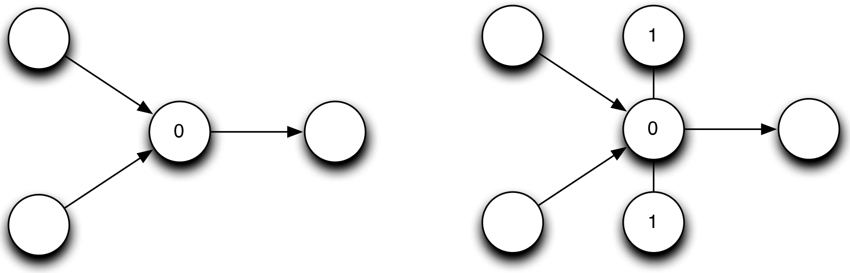

Fig. 4. Gates with diodes.

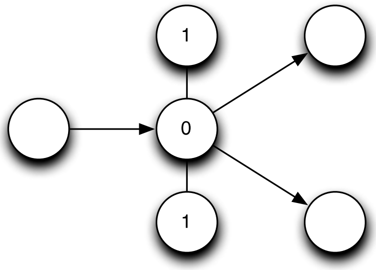

Fig. 5. OR gate 'backwards' to multiply the information of an output.

Then, given ϕ a monotone Boolean circuit and an input I, we can build a graph G that simulates ϕ in I. Setting active the input nodes that are true, and passive the rest, and considering v as the output of the circuit, $(G, x(0), v)$ belongs to **PER** if and only if (ϕ, I) belongs to **M2CVP**.

This reduction can be done by a Turing machine in logarithmic space: reading the input (a monotone circuit) of size n, the machine only has to determine which construction corresponds to each gate ($\mathcal{O}(1)$ space), and then determine the connectivity of every gate ($\mathcal{O}(\log n)$ space). Then, the whole construction requires $\mathcal{O}(\log n)$ space. $\square$

Remark 1. Note that in the construction of the OR gate and the diode, we need vertices with degree 5. We will show that a construction that uses vertices with degree less than 5 is not possible unless **P** = **NC**.

3. Case $\Delta(G) \leq 4$

To prove the second part of the theorem, it will be useful to work with the vertices that are initially passive, so we will call $G[0] = (V[0], E[0])$ the induced subgraph of initially passive vertices of G:

$$V[0] = \{u \in V \mid x_u(0) = 0\}$$
$$E[0] = \{uw \in E \mid u, w \in V[0]\}.$$

Definition 1. A community in G is a subset of nodes $X \subset V$ each of which has at least as many neighbors in X as in $V \backslash X$, i.e. for every $v \in X$, $|N(v) \cap X| \geq |N(v) \cap (V \backslash X)|$.

Definition 2. Given an initial configuration $x(0)$, we say that a passive vertex v is *stable* if $x_v(t) = 0\ \forall t > 0$.

Remark 2. A vertex is stable if and only if it belongs to a community X of initially passive vertices.

Remark 3. In the case of $\Delta(G) \leq 4$, every cycle is a community, so every cycle in $G[0]$ is stable.

Notice that even if G is connected, $G[0]$ may not be. Given $v \in V$, let $C[v]$ the connected component of v in $G[0]$, and $G[0, v] = (C[v], E[0, v])$ the subgraph of G induced by $C[v]$, that is:

$$E[0, v] = \{uw \in E \mid u, w \in C[v]\}.$$

Lemma 2. *Let G with $\Delta(G) \leq 4$. A vertex v is stable if and only if there exists a path P of $G[0, v]$ that contains v and, if u is one of its ends, then*

1. *u belongs to a cycle in $G[0]$, or*
2. *$d_G(u) \leq 2$.*

Proof. Suppose first that v is stable. We know that at most after $T = |V|$ steps, the dynamic given by x gets into a fixed point, this is $x(T + t) = x(T)\ \forall t \geq 0$. Let $G'[0, v]$ the connected component of passive vertices of G in the step T that contains v. Notice that $G'[0, v]$ is a subgraph of $G[0, v]$. Take P as the longest path in $G'[0, v]$ that contains v. Let u be an end of P. Suppose first that u has only one neighbor in $G'[0, v]$, since u is stable, u has at most one more neighbor in G, hence $d_G(u) \leq 2$. Suppose now that u has more than one neighbor in $G'[0, v]$. Since P is the longest path that contains v in $G'[0, v]$, both neighbors of u must belong to P. Then u belongs to a cycle in $G[0]$.

Let us prove the converse. Let P be a path in $G[0, v]$, with $v \in P$ and its ends satisfying (1) or (2). Let X be the set of vertices of P, plus, for each end-point u of P satisfying condition (1), the corresponding cycle C_u of $G'[0, v]$ containing u. Note that, since $\Delta(G) \leq 4$, X forms a community of initially passive vertices. Indeed, each interior vertex of P (P minus its ends) has at least half of its neighbors in X, and the same holds for the vertices of the cycles added to X. If an endpoint u of P satisfies condition (2), it also has at least half of its neighbors in X, thus X is a community. The proof follows from Remark 2. □

In the introduction we mentioned that finding connected and biconnected components of a given graph can be done in **NC**, both of them in $\mathcal{O}(\log^2 n)$ time, using a total of $\mathcal{O}(n^2)$ processors. The proof of these propositions, which consists in fast parallel algorithms, can be found in [12]. In order to make use of these results, we will need to know the inputs and the outputs of these algorithms, detailed in Algorithms 1 and 2.

```
Algorithm 1 Connected components
Require: The $n \times n$ adjacency matrix $A$ of an undirected graph.
Ensure: An array $D$ of size $n$ such that $D(i)$ is equal to the smallest vertex in the connected component of $i$.
```

```
Algorithm 2 Biconnected components
Require: The $n \times n$ adjacency matrix $A$ of an undirected connected graph.
Ensure: An array $B$ such that $B(e) = B(g)$ if and only if $e$ and $g$ are in the same biconnected component.
```

In some steps of our algorithm, we will need to compute the prefix sum of an array. Given a group $G = (X, +)$, the prefix sum of a sequence of numbers $x_0, x_1, \ldots x_n \in X$ is a second sequence of numbers $y_0, y_1, \ldots, y_n \in X$, where $y_k = \sum_{i=0}^{k} x_i$. JáJá [12] provides fast parallel algorithms that uses $\mathcal{O}(\log n)$ time and $\mathcal{O}(n)$ processors to calculate the prefix sum of a given set.

Now we are ready to prove the second part of the Theorem 1. (2):

Proof. From Lemma 2, to decide **PER** it is enough to determine if v belongs to a path in $G[0, v]$ with ends satisfying (1) or (2). This must be done quickly in parallel, so we can not apply regular search methods like Depth-First Search to achieve this.

We can obtain $G[0, v]$ quickly in parallel with Algorithm 1 and a prefix sum algorithm. First, to build $G[0]$, we calculate its adjacency matrix $A[0]$ from A and the initial configuration $x(0)$: letting $\bar{x} = 1 - x(0)$ and then applying the prefix sum algorithm to it, we obtain a vector s such that s_n will be the number of initially passive vertices, and for any i, j such that $x_i = x_j = 0$, if $A_{i,j} = 1$ then $A[0]_{s_i s_j} = 1$. This can be done in $\mathcal{O}(1)$ time using $\mathcal{O}(n^2)$ processors.

Then, with Algorithm 1, we calculate the connected components of $G[0]$ in $\mathcal{O}(\log^2 n)$ time and $\mathcal{O}(n^2)$ processors. With the array given by Algorithm 1, we build $A[0, v]$ the adjacency matrix of $G[0, v]$. Notice that if v is isolated in $G[0, v]$, then v is stable. Suppose now that v is not isolated in $G[0, v]$.

A biconnected component that contains more than two vertices has the property that any two vertices are on a same cycle, and conversely the vertices of any cycle are in a same biconnected component. It follows that from the array given by Algorithm 2 we can obtain which vertices of $G[0, v]$ satisfy (1). Using Algorithm 2, this can be done quickly in parallel.

To calculate the vertices that satisfy (2) we obtain from A the vector D of dimension $|V|$, where $D_u = d_G(u)$. This can be done in $\mathcal{O}(\log n)$ time and $\mathcal{O}(n)$ processors with a prefix sum algorithm. Then, any vertex u of $G[0, v]$ where $D_u \leq 2$ satisfies (2).

To determine if v belongs to a path with ends satisfying (1) or (2), we build another graph $\overline{G} = (\overline{V}, \overline{E})$ from $G[0, v]$ and a new vertex called ∞, where:

$$\overline{V} = V[0, v] \cup \{\infty\}$$
$$\overline{E} = E[0, v] \cup \{u\infty \mid u \in V[0, v] \text{ satisfying (1) or (2)}\}.$$

Notice that v is stable if and only if there are two different paths from v to ∞. Then, v is stable if and only if there is a cycle in $\overline{G}$ that contains v and ∞. In conclusion, the last steps of the algorithm must calculate the biconnected components of $\overline{G}$, and then decide if v and ∞ are in the same component.

The result is Algorithm 3.

The correctness of the algorithm is obtained as follows: if v is stable, then by Lemma 2, v belongs to a path P in $G[0, v]$ whose ends satisfy (1) or (2). By definition of $\overline{A}$, the ends of P are connected to the vertex ∞, then, there exists a cycle C that contains v and ∞. Conversely, if there is a cycle in $\overline{A}$ that contains v and ∞, then by definition of $\overline{A}$, there is a path P in $G[0, v]$ whose ends satisfy (1) or (2), and then, by Lemma 2, v is stable. In the Appendix there is a deep analysis of the complexity of this algorithm, which concludes that it can be done in a PRAM machine in $\mathcal{O}(\log^2 n)$ time, with $\mathcal{O}(n^4)$ processors. □

Algorithm 3 PER

```
Require: The $n \times n$ adjacency matrix $A$ of a undirected graph $G = (V, E)$, an initial configuration $x(0)$ and a vertex $v \in V$
Ensure: Accept if $\{G, x(0), v\}$ belongs to PER and Reject otherwise.
 1: Calculate $A[0]$, the adjacency matrix of $G[0]$ using $x(0)$ and a prefix sum algorithm.
 2: Calculate $A[0, v]$, the adjacency matrix of $G[0, v]$, using $A[0]$ in Algorithm 1
 3: if $v$ is isolated in $A[0, v]$ then
 4:     Accept, and quit.
 5: end if
 6: Calculate $B$ the output of Algorithm 2 in $A[0, v]$.
 7: Calculate the cycles in $A[0, v]$ from $B$ and store in a Boolean array $C$ where $C_u = 1$ iff $u$ belongs to a cycle in $A[0, v]$.
 8: Calculate $D$, the vector of degree of $G$.
 9: Define $\overline{A}$, the adjacency matrix of $\overline{G}$, using $A[0, v]$, $D$ and $C$.
10: Calculate $\overline{B}$ the output of Algorithm 2 in $\overline{A}$.
11: Using $B$ determine if $v$ and $\infty$ are in a cycle. Reject if they do, Accept in other case.
```

4. Related problems

We consider here two other variants of dynamics, the simple (non strict) majority voting and a dynamic with a fixed threshold.

4.1. Simple majority

Consider the rule given by the following transition function:

$$f_i(x) = \begin{cases} 1 & \text{if } x_i = 1 \\ 1 & \text{if } \displaystyle\sum_{j \in N(i)} x_j \geq \frac{|N(i)|}{2} \text{ and } x_i = 0 \\ 0 & \text{if } \displaystyle\sum_{j \in N(i)} x_j < \frac{|N(i)|}{2} \text{ and } x_i = 0. \end{cases}$$

We call it *the simple majority rule* and we have a similar result:

Theorem 3. *For the simple majority rule:*

1. *On the family of graphs $G = (V, E)$ such that $\Delta(G) \geq 4$, the problem* **PER** *is* ***P****-Complete.*
2. *On the family of graphs $G = (V, E)$ such that $\Delta(G) \leq 3$, the problem* **PER** *is in* ***NC****.*

For the proof of this theorem we will need a version of Lemma 2 for this rule.

Lemma 4. *Let G with $\Delta(G) \leq 3$. A vertex v is stable for the non strict majority if and only if there exists a path P of $G[0, v]$ that contains v and, if u is one of its ends, then*

1. *u belongs to a cycle in $G[0]$, or*
2. *$d_G(u) = 1$.*

The proof of this lemma is analogous to the proof of Lemma 2.

Proof of Theorem 3. 1. Fig. 6 show the gadgets used to simulate a monotone circuit with the non strict majority rule. Notice that in this case we only need vertices with degree 4.
2. For the second part, we consider Lemma 4 and modify Algorithm 3 only in the definition of $\overline{A}$: We connect to ∞ vertices in $G[0, v]$ with degree 1 and vertices that belong to cycles in $G[0, v]$. □

4.2. θ-Rule

Given $\theta > 0$ and $G = (V, E)$ such that $\delta(G) \geq \theta$ (i.e. $d_G(v) \geq \theta \ \forall v \in V$). Consider the rule given by the following transition function:

$$f_i(x) = \begin{cases} 1 & \text{if } x_i = 1 \\ 1 & \text{if } \displaystyle\sum_{j \in N(i)} x_j > \theta \text{ and } x_i = 0 \\ 0 & \text{if } \displaystyle\sum_{j \in N(i)} x_j \leq \theta \text{ and } x_i = 0. \end{cases}$$

We call it *the θ-rule* and the result in this case is:

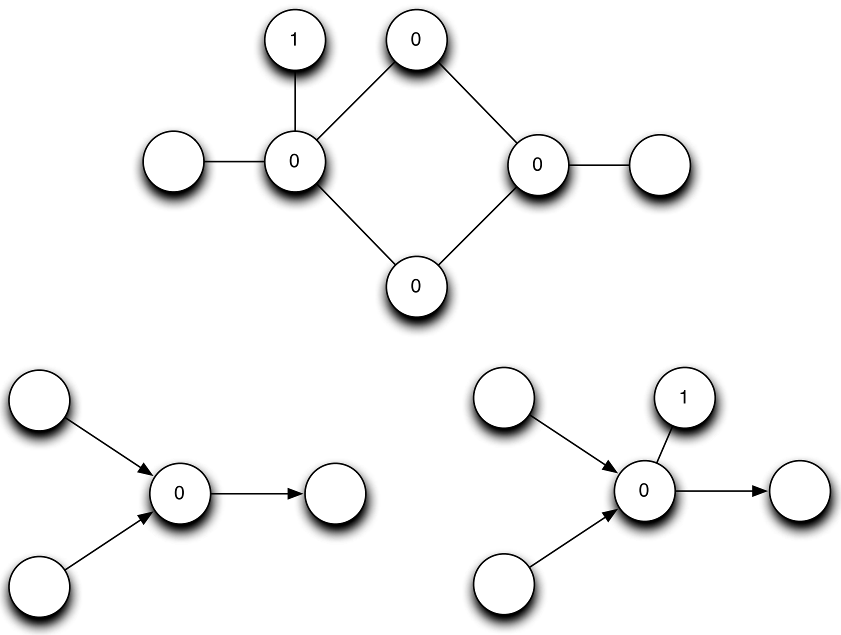

Fig. 6. Top: the diode for the non strict case, bottom: the AND and OR gates for the same rule.

Theorem 5. *For the θ-rule:*

1. *On the family of graphs $G = (V, E)$ such that $\Delta(G) \geq \theta + 3$, the problem* **PER** *is* ***P****-Complete.*
2. *On the family of graphs $G = (V, E)$ such that $\Delta(G) \leq \theta + 2$, the problem* **PER** *is in* ***NC****.*

Remark 4. Notice that in this case, a passive node becomes active if it has at least $\theta + 1$ active neighbors. Then, in the case of $\Delta(G) \leq \theta + 2$, a cycle in $G[0]$ will be stable.

For the proof of Theorem 5 we will also need a version of Lemma 2 for this rule.

Lemma 6. *Let G with $\Delta(G) \leq \theta + 2$. A vertex v is stable for the non strict majority if and only if there exists a path P of $G[0, v]$ that contains v and, if u is one of its ends, then*

1. *u belongs to a cycle in $G[0]$, or*
2. *$d_G(u) \leq \theta + 1$.*

The proof of this lemma is analogous to the proof of Lemma 2.

Proof of Theorem 5. 1. Like in the other cases, we will simulate a monotone Boolean circuit with a graph G. Since $\delta(G) \geq \theta$, we have to change the constructions of Theorem 1(1), in order to obtain vertices with degree at least θ. The solution is to take the gadgets of Theorem 1.(1), and connect the vertices to p other active vertices, that are connected with each other, where p is some convenient number (obviously depending on θ). In other words, we connect every vertex of the gadgets to every vertex of a complete graph of active vertices.

Fig. 7 shows how to simplify the notation. We represent the whole complete subgraph K and the $|K|$ connections by a squared vertex with the number $|K|$ inside.

Using this, it is easy to build the new gadgets, shown in Fig. 8. Notice that in the construction of the OR gate and the diode, the degree required is $\theta + 3$.

2. In this part, we consider Lemma 6 and modify Algorithm 3 only in the definition of $\overline{A}$: We connect to ∞ vertices in $G[0, v]$ with degree $\leq 1 + \theta$ and vertices that belong to cycles in $G[0, v]$. $\square$

5. Conclusion

We have discussed several variants of majority voting rules on Boolean automata networks, and we have shown that the complexity of the bootstrap percolation problem depends on the maximum degrees of the input graphs. When this maximum degree is small, the problem is in **NC**, and for large degrees the problem becomes **P**-Complete. Clearly, our bounds are tight. Our **P**-completeness proofs use a reduction from a special case of the **CVP** problem, which is **P**-Complete on arbitrary circuits.

A natural extension of our results would be to consider special graph classes.

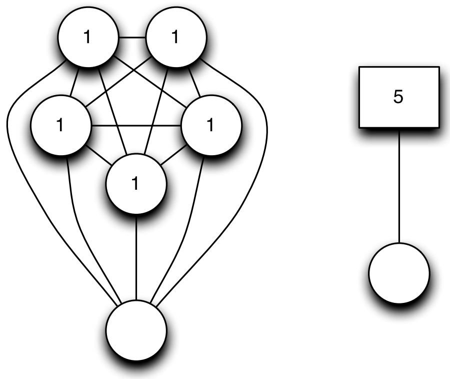

Fig. 7. We represent a complete subgraph K, in this case K_5, and the $|K|$ connections with a single squared vertex with the number $|K|$ inside with one edge.

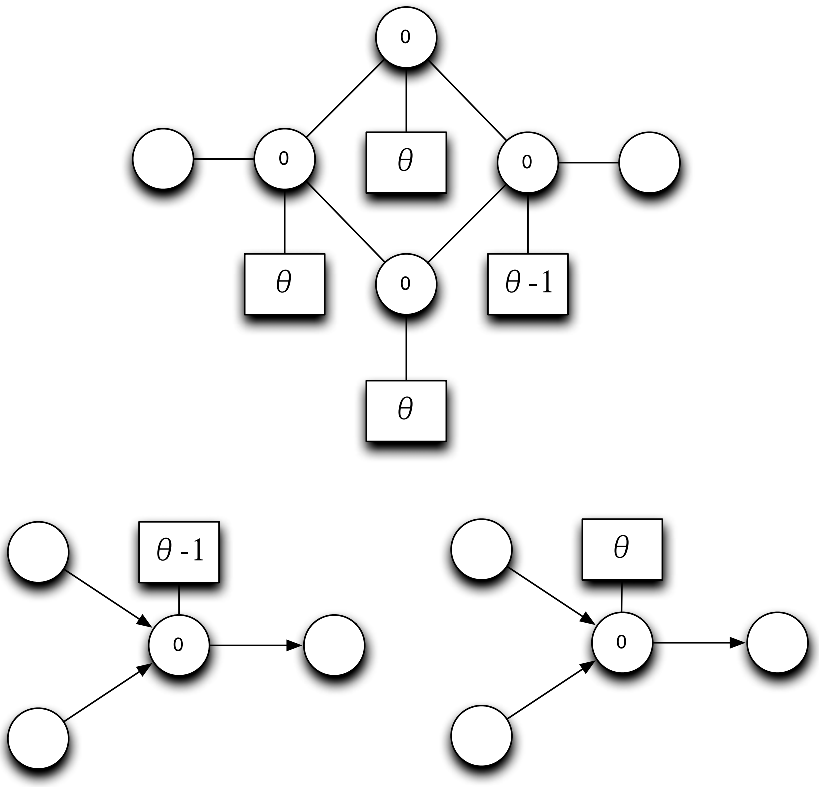

Fig. 8. Top: the diode for the θ rule, bottom: the AND and OR gates for the same rule.

A straightforward question is: What happens with bootstrap percolation **PER** in the planar case? It is obvious that the results for $\Delta(G) \leq 4$ remain true. So we are thinking of the case when the max degree is at least 5. We point out that monotone circuit value problem **MCVP** becomes **NC** when restricted to planar Boolean circuits [13], so we cannot use our reduction technique for the planar case.

Acknowledgments

The first and second authors' work was partially supported by Fondecyt 1100003 (Eric Goles, Pedro Montealegre-Barba), BASAL-CMM (Eric Goles), and Anillo ACT-88 (Eric Goles, Pedro Montealegre-Barba).

The author's research was partially done while visiting the LIFO, Université d'Orléans.

Appendix. Analysis of the complexity of Algorithm 3

Now we will give a detailed analysis of the complexity of Algorithm PER, step by step:

Step 1: Calculate $A[0]$, the adjacency matrix of $G[0]$ using $x(0)$ and a prefix sum algorithm.

1. We first calculate $\overline{x(0)}$ where
$$\overline{x_i(0)} = \begin{cases} 0 & \text{if } x_i(0) = 1 \\ 1 & \text{if } x_i(0) = 0. \end{cases}$$
This can be done in $\mathcal{O}(1)$ time with $\mathcal{O}(n)$ processors.
2. Then we use the prefix sum algorithm on $\overline{x(0)}$ to calculate an array s, where
$$s_i = \sum_{j=0}^{i} \overline{x_j(0)}$$
then s represents the new labels of the vertices of G in the graph $G[0]$, and s_n equals the number of vertices of $G[0]$. This can be done in $\mathcal{O}(\log n)$ time with $\mathcal{O}(n)$ processors.
3. With s, $x(0)$ and A, build $A[0]$, the adjacency matrix of $G[0]$: The dimensions of $A[0]$ are $s_n \times s_n$. For any i, j such that $x_i(0) = x_j(0) = 0$, if $A_{ij} = 1$ then $(A[0])_{s_i s_j} = 1$, otherwise $(A[0])_{s_i s_j} = 0$. This can be done in $\mathcal{O}(1)$ time with $\mathcal{O}(n^2)$ processors.

Step 2: Calculate $A[0, v]$, the adjacency matrix of $G[0, v]$, using $A[0]$ in Algorithm 1.

1. First calculate C, the output of Algorithm 1 on input $A[0]$. This can be done in $\mathcal{O}(\log^2 n)$ time with $\mathcal{O}(n^2)$ processors.
2. Remember that s_v is the label of v in $A[0]$. Then C_{s_v} is the label of the connected component that contains v. Calculate $\overline{C}$, an array with the same dimensions as C such that $\overline{C}_i = 1$ if $C_i = C_{s_v}$. This can be done in $\mathcal{O}(1)$ time with $\mathcal{O}(n)$ processors.
3. Use the prefix sum algorithm on $\overline{C}$, obtaining S, where
$$S_i = \sum_{j=0}^{n} \overline{C}_j$$
then S represents the new labels of the vertices of $G[0]$ in the graph $G[0, v]$, and S_n equals the number of vertices of $G[0]$. This can be done in $\mathcal{O}(\log n)$ time with $\mathcal{O}(n)$ processors.
4. With S, $\overline{C}$ and $A[0]$, build $A[0, v]$, the adjacency matrix of $G[0, v]$: The dimensions of $A[0, v]$ are $S_n \times S_n$, and if $A[0]_{ij} = 1$, with $\overline{C}_i = \overline{C}_j = 1$, then $(A[0, v])_{S_i S_j} = 1$, and $(A[0, v])_{S_i S_j} = 0$ in the other case. This can be done in $\mathcal{O}(1)$ time with $\mathcal{O}(n^2)$ processors.

Step 3,4,5: Determine if v is isolated; this can be done in $\mathcal{O}(1)$ time with $\mathcal{O}(n)$ processors.
Step 6: Calculate B the output of Algorithm 2 in $A[0, v]$. This can be done in $\mathcal{O}(\log^2 n)$ time with $\mathcal{O}(n^2)$ processors.
Step 7: Calculate the cycles in $A[0, v]$ from B and save it in an array C, where $C_u = 1$ if u belongs to a cycle in $A[0, v]$.

1. Calculate M, the maximum value in B and N the length of B. This can be done in $\mathcal{O}(\log n)$ time with $\mathcal{O}(n)$ processors. Note that $M, N \le n^2$.
2. Compute the matrix K of dimensions $M \times N$, where $K_{ij} = 1$ if $B_j = i$. This can be done in $\mathcal{O}(1)$ time with $\mathcal{O}(n^4)$ processors.
3. Calculate the array L, where $L_i = \sum_{j=1}^{N} K_{ij}$, i.e. L_i is the number of edges in component i. This can be done in $\mathcal{O}(\log n)$ time with $\mathcal{O}(n^2)$ processors.
4. Compute the array C: let $g = uv$, (i.e. $A[0, v]_{u,v} = 1$) with $B_g = k$ and $L_k \ne 1$ then $C_u = C_v = 1$. This can be done in $\mathcal{O}(1)$ time with $\mathcal{O}(n^2)$ processors.

Step 8: Calculate D, the vector of degrees of $\underline{G}$. This can be done in $\mathcal{O}(\log n)$ time with $\mathcal{O}(n^2)$ processors.
Step 9: Define $\overline{A}$, the adjacency matrix of $\overline{G}$, using $A[0, v]$, D and C. Let m the dimension of $A[0, v]$. $\overline{A}$ has dimensions $m + 1 \times m + 1$, because $\overline{A}$ is the same as $A[0, v]$ with one more row and column, corresponding to the vertex ∞. Then, to calculate $\overline{A}$, we copy $A[0, v]$ and $\overline{A}_{i,m+1} = \overline{A}_{m+1,i} = 1$ if $D_i \le 2$ or $C_i = 1$. This can be done in $\mathcal{O}(1)$ time with $\mathcal{O}(n^2)$ processors.
Step 10: Calculate $\overline{B}$ the output of Algorithm 2 in $\overline{A}$. This can be done in $\mathcal{O}(\log^2 n)$ time with $\mathcal{O}(n^2)$ processors.
Step 11: Using B, determine if v and ∞ are in a cycle.

1. Calculate M, the maximum value in B and N the length of B. This can be done in $\mathcal{O}(\log n)$ time with $\mathcal{O}(n)$ processors. Note that $M, N \le n^2$.
2. Compute arrays I, V with length M: Let $g \in E(\overline{G})$, with $B(g) = k$. If $\infty \in g$, then $I_k = 1$, and if $v \in g$, then $V_k = 1$. This can be done in $\mathcal{O}(1)$ time with $\mathcal{O}(n^2)$. Notice that V_k (resp. I_k) is 1 if v (resp. ∞) is in a edge in component k.
3. Calculate the matrix K of dimensions $M \times N$, where $K_{ij} = 1$ if $B_j = i$. This can be done in $\mathcal{O}(1)$ time with $\mathcal{O}(n^4)$ processors.
4. Compute the array L where $L_i = \sum_{j=1}^{N} K_{ij}$, i.e. L_i is the number of edges in component i. This can be done in $\mathcal{O}(\log n)$ time with $\mathcal{O}(n^2)$ processors.
5. Calculate $I + V$. If for some k, $(I + V)_k \ge 2$ and $L_k > 1$ then v and ∞ are in the same (non trivial) component, and so in a cycle.

In conclusion, the whole algorithm can be executed in $\mathcal{O}(\log^2 n)$ time, with $\mathcal{O}(n^4)$ processors.

References

[1] J. Chalupa, P.L. Leath, G.R. Reich, Bootstrap percolation on a Bethe lattice, Journal of Physics C 12 (1979) L31–L35.
[2] J. Balogh, G. Pete, Random disease on the square grid, Random Structures & Algorithms 13 (1998) 3–4.
[3] M. Treaster, W. Conner, I. Gupta, K. Nahrstedt, Contagalert: Using contagion theory for adaptive, distributed alert propagation, in: NCA'06, 2006, pp. 126–136.
[4] S. Manna, Abelian cascade dynamics in bootstrap percolation, Physica A: Statistical Mechanics and its Applications 261 (3–4) (1998) 351–358. http://dx.doi.org/10.1016/S0378-4371(98)00346-X.
[5] R.A. Meyers, Encyclopedia of Complexity and Systems Science - v. 1-10, 1st Edition, Springer Publishing Company, Incorporated, 2009.
[6] I. Rapaport, K. Suchan, I. Todinca, J. Verstraete, On dissemination thresholds in regular and irregular graph classes, Algorithmica 59 (2011) 16–34. http://dx.doi.org/10.1007/s00453-009-9309-0.
[7] R. Carvajal, M. Matamala, I. Rapaport, N. Schabanel, Small alliances in graphs, in: MFCS'07, 2007, pp. 218–227.
[8] C. Moore, Majority-Vote cellular automata, Ising dynamics, and P-completeness, Journal of Statistical Physics 88 (1997) 795–805.
[9] C. Moore, M.G. Nordahl, Predicting lattice gases is P-complete, Santa Fe Institute Working Paper 97-04-034.
[10] D. Griffeath, C. Moore, Life without death is P-complete, Complex Systems 10 (1997) 437–447.
[11] R. Greenlaw, H. Hoover, W. Ruzzo, Limits to Parallel Computation: P-completeness Theory, Oxford University Press, 1995.
[12] J. JáJá, An Introduction to Parallel Algorithms, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1992.
[13] A.L. Delcher, S.R. Kosaraju, An nc algorithm for evaluating monotone planar circuits, SIAM Journal on Computing 24 (1995) 369–375. http://dx.doi.org/10.1137/S0097539792226278.