Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n° 8512

Leader Election without Compass in Some Hyperbolic and Euclidean Cellular Automata

Codrin Nichitiu
Christophe Papazian
Eric Rémila

February 2001

Research Report N° 2001-08

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr

Leader Election without Compass in Some Hyperbolic and Euclidean Cellular Automata

Codrin Nichitiu
Christophe Papazian
Eric Rémila

February 2001

Abstract

We present a linear time algorithm for the networking and distributed computing problem of leader election (LE). Given a graph, its vertices represent processors (here finite state machines), and its edges communication lines (here synchronous). The LE problem consists in finding a protocol for a family of graphs such that after iterating it, a vertex, edge or cycle be distinguished by a special state called leader. Here the graphs are only required to be connected, and without holes. We describe the algorithm in full detail on a special class of planar graphs, prove its correctness and show how it extends to other classes.

Keywords: Distributed Algorithms, Graph Automata, Leader Election, Euclidean Cellular Automata, Hyperbolic Cellular Automata

Résumé

Nous présentons un algorithme en temps linéaire, pour résoudre le problème de l'élection d'un chef sur un graphe dont chaque sommet est un automate fini (un ordinateur) et chaque arête, un lien de communication (modèle synchrone). Le problème de l'élection consiste à trouver un algorithme pour une famille de graphe qui, après un nombre fini de calcul, distingue un sommet, une arête ou un cycle particulier dans un état de "chef". Ici les graphes considérés doivent être connexes et sans trous. Nous décrivons l'algorithme en détail sur une classe de graphes particulière, prouvons son exactitude et montrons comment il s'étend aux autres classes.

Mots-clés: Algorithmes distribués, Graphes d'automates, Election, Automate Cellulaire Euclidien, Automate Cellulaire Hyperbolique

Leader Election without Compass in Some Hyperbolic and Euclidean Cellular Automata

Codrin Nichitiu*, Christophe Papazian, and Eric Rémila

*EURISE, Fac. Sci. Tech., Univ. Jean-Monnet, 23, rue du Dr. Paul Michelon, 42023 ST ETIENNE, France
LIP, École Normale Supérieure de Lyon, 46, allée d'Italie, 69364 LYON Cedex 07, France
e-mail: Codrin.Nichitiu@univ-st-etienne.fr,
Christophe.Papazian@ens-lyon.fr,
Eric.Remila@ens-lyon.fr
Fax : +33 4 72 72 80 80

Abstract We present a linear time algorithm for the networking and distributed computing problem of leader election (LE). Given a graph, its vertices represent processors (here finite state machines), and its edges communication lines (here synchronous). The LE problem consists in finding a protocol for a family of graphs such that after iterating it, a vertex, edge or cycle be distinguished by a special state called leader. Here the graphs are only required to be connected, and without holes. We describe the algorithm in full detail on a special class of planar graphs, prove its correctness and show how it extends to other classes.
Keywords: Distributed Algorithms, Graph Automata, Leader Election, Euclidean Cellular Automata, Hyperbolic Cellular Automata

1 Introduction

The problem we are here interested in is the leader election problem, well known from the network and distributed computing research. This problem consist in finding, through a local process, a vertex which can be distinguished for further (synchronized) computations. The process has to be local and fundamentally the same for a whole family of graphs describing the network.

Here the leader election is studied on CA (i.e. the evolution has synchronous time steps, and the size of the memory of each node is bounded), with a planar architecture, called Graph Automata. Several algorithms have already been devised for this framework (as in [4], [1], [7]), but these authors only work on subgraphs of $\mathbb{Z}^2$, and also assume that each node can use a compass (i.e. can determine the North, South, West and East directions, using thus implicitly the coordinate system).

In this paper, we present an efficient algorithm which does not use a compass: we only assume that each node can recognize its left side from its right side. In terms of global information, the left-right condition is weaker than the compass condition. In this way, election can also be achieved in architectures constructed of hyperbolic tessellations of the plane. However, we also have to indicate a drawback: the algorithm requires the absence of holes. To the opposite, for example, the algorithm in [4] also works for graphs with holes.

In Section 2 we give the necessary definitions, and we precisely state the result; in Section 3 we describe the algorithm, in Section 4 we outline its correctness proof, and in Section 6 we explain its extensions to multi-cardinality-face planar graphs and higher dimensions.

2 Definitions and result

2.1 Regular planar tiling graphs, dual graphs

A (finite) graph G is a set $G = (V, E)$ with V a (finite) set of vertices, and E a subset of V^2, whose elements are called "edges". We will only consider graphs without loops (no (v, v) edges) and symmetric (if (v, w) is in E, so is (w, v)). A vertex w is a neighbor of another vertex v if (v, w) is in E. The degree of a vertex is the number of its neighbors; the degree of the graph is the maximum degree among its vertices. A graph is connected when for any $x, y \in V$ there exists a $x - y$ path, i.e. the vertices $z_1, z_2, \ldots, z_n \in V$ with $(x, z_1), (z_i, z_{i+1}), (z_n, y) \in E$, where $1 \leq i < n$. If $x = y$, the $x-y$ path is called a cycle, and if $x \neq z_i \neq z_j$ for $i \neq j$, then the cycle is elementary. A chord is an edge linking two non-consecutive vertices of a cycle (in its enumeration). A chordless elementary cycle of at least three vertices is called a face.

A planar graph is, informally speaking, a graph which can be drawn on a plane (Euclidean or not) without any crossing of its arcs. A subgraph $G' = (V', E')$ of a graph $G = (V, E)$ has $V' \subset V$, and $E' \subset E \cap (V' \times V')$. G' is a FCWHS (finite, connected and without holes) when it is connected, it is such that its complement (with respect to G) is connected and has $|V'|$ finite. A non-connected graph G is a union of components, each being a maximal connected subgraph.

We denote by $\Gamma[k, d]$ a locally finite connected planar graph, uniquely defined by the couple (k, d), regular of degree d, with all its faces of length k. These graphs can be seen as finite regular tessellations (see [2]) of the spherical plane, for $1/k+1/d \geq 1/2$, or as infinite regular tessellations of the Euclidean plane, for $1/k + 1/d = 1/2$, and of the hyperbolic plane, for $1/k + 1/d \leq 1/2$, the later being the only set having an infinite number of different graphs.

Given a planar representation of a planar graph $G = (V, E)$, let the interior dual G^* be the graph (V^*, E^*) such that V^* is the set of faces of G (minus the exterior face if G is finite) and that for any two elements f, f' of V^*, faces f and f' share an edge in G if and only if $(f, f') \in E^*$. Hence, $(\Gamma[k, d])^* = \Gamma[d, k]$.

We thus prevent the interior dual from including the "border" face (that has infinite "surface", when it exist), having in turn the possibility that this interior dual is not always connected, even when G is connected.

For a graph $G = (X_A, E_A)$, we also define the superposition graph G_{e} where G^* is its interior dual $G^* = (X_{G^*}, E_{G^*})$.

$$G_{\mathrm{e}} = (X = X_G \cup X_{G^*}, E = E_G \cup E_{G^*} \cup D(G, G^*))$$

where $D(G, G^*)$ is the set of possible mixed edges linking a vertex and its dual:

$$D(G, G^*) = \{(v, f) | v \in X_G, f \in X_{G^*} \text{ and } v \text{ is a vertex of the face } f\}.$$

2.2 Graph automata

Let d be a fixed integer such that $d \geq 2$. A d-graph is a couple (G, g), where $G = (V, E)$ is a symmetric connected graph with only two kinds of vertices : vertices of degree 1 (which are called #-vertices by Rosenfeld) and vertices of degree d. g is a mapping from E to $\{1, 2, ..., d\}$ such that, for each d-vertex v of V, the partial mapping $g(v, .)$ is injective. Let $\widehat{g}(v, i) = x$ when $g(v, x) = i$. For planar graphs, we also require $g(v, .)$ to be homogeneous, that is when drawing the graph, the circularly consecutive edges around each vertex have consecutively increasing labels (modulo d) counterclockwise. This actually amounts to the notion of "left"-"right" recognition from local information.

A finite d-automaton is a pair (Q, δ) such that Q is a finite set of states with # in Q, and δ is a function from $Q \times (\mathbb{Z}_d)^d \times Q^d$ to Q such that, for each element Υ of $Q \times (\mathbb{Z}_d)^d \times Q^d$, $\delta(\Upsilon) = \#$ if and only if the first component of Υ is #.

A graph automaton is a couple $M = (G_d, A)$ with G_d a d-graph, and A a finite automaton (Q, δ). A cellular automaton is a graph automaton over $\mathbb{Z}^d$ or a Cayley graph. A configuration C of M is a mapping from the set of vertices of G_d to the set of states Q, such that $C(v) = \#$ iff v is a #-vertex. We compute a new configuration from a previous one by applying the transition function δ simultaneously to each vertex of G_d, computing a new state for each (non-#) vertex by reading its state and the states of its neighbors, using the vector of neighborhood $H(v) = (g(\widehat{g}(v, 1), v), \ldots, g(\widehat{g}(v, d), v))$:

$$C_{new}(v) = \delta(C(v), H(v), C(\widehat{g}(v, 1)), \ldots, C(\widehat{g}(v, d)))$$

(#-vertices always stay in the # state). Hence, we have a synchronous model, with local and finite memory.

Remark 1. The labeling functions are also needed because the flow of information between the vertices needs to be directed, that is the vertices have to distinguish their neighbors, by knowing how they are labeled by their very neighbors. They have been introduced by Wu and Rosenfeld in [8], and proven to be necessary in [3].

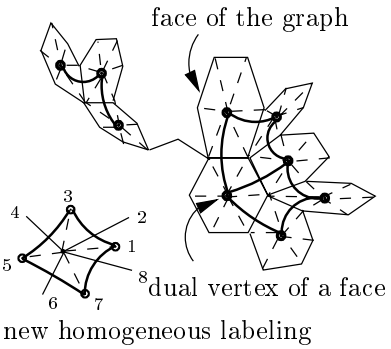

Figure1. Adding dual vertices of the faces, able to communicate with their neighbors and with the vertices of their dual faces.

2.3 Election problem

Given a class of d graphs, the problem is to find a finite set of states Q and a transition function such that starting with the configuration having all vertices in the same state (or in a state from a special subset) called a start state, after a finite number of iteration steps depending on the graph, the final configuration is such that

– one vertex is in a special state called leader
– all the other vertices are in special states called soldiers

We extend this definition by allowing two or k vertices (forming an edge, respectively a face) to be elected, since in some graphs, for symmetry reasons, we cannot always isolate one last vertex locally (see Subsection 3.1). Thus we can say that at the end of the process a vertex, an edge or a face must be elected. This does not cause fundamental problems, since the vertices are neighbors and synchronously elected, thus "aware" of the "power-sharing" situation.

2.4 Results

We give here an automaton for the election on FCWHS of $\Gamma[k,d]$, for any k and d (such that $\Gamma[k,d]$ is infinite), homogeneously labeled (see subsection 2.2). In order to ease the presentation, we explain it on the superposition graph G_e; this is a mere technical artifact, as it is explained in Section 5 (also see [6] and [5]). This algorithm can easily be extended as it is explained in Subsection 6.

3 Algorithm

Before detailing the general election algorithm, we need to explain a very simple one, that is the election on trees.

3.1 Tree election

The idea is to start from the leaves, and to iteratively say that

1. all leaves are soldiers
2. any soldier is no longer part of the tree
3. an isolated vertex is the leader.

This has to be slightly changed yet, to deal with parity problems. Thus an intermediate state is introduced, giving the finite automaton in the figure 2. Note that either a vertex or an edge is elected.

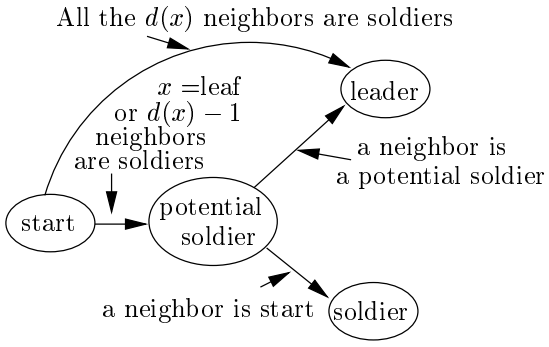

Figure2. Transitions of the finite state automaton placed in the vertex x, of degree $d(x)$, of the tree.

3.2 Election on the superposition $G + G^*$

Overview

Now, we can try to make use of the previous algorithm through what might be seen as the "skeleton" graph of the initial tiling graph. Each subgraph only composed of faces, linked to other such subgraphs through one-vertex-large paths, is iteratively

shrinked until it becames a vertex. Then, that part of the graph becomes isomorphic to a tree, and thus the tree-election algorithm can be applied (see figure 5).

The dual G^* is used to simplify the shrinking process. At the beginning, the active vertices (those which initiate the election process) form a sort of "tree of cycles". We have to say

1. how the tree of cycles is created at the beginning
2. how the cycles are shrinking
3. what happens when they collapse to one point
4. how the active vertices always form a connected graph

Initialization

For a border vertex there are several possible cases, because it vertex can belong to faces or not, and it also can have or not all its neighbors in start state.

The initialization takes k time steps: each vertex explores the k-vertex faces to which it belongs, and decides according to their answer.

Remember the labeling is homogeneous. All start-state vertices launch signals which should go round each face to which they belong, all in the same way (say clockwise). This is very easy to implement, through for example a "label stack signal", popped each time it reaches a vertex, and directed through the top label to the subsequent vertex on the cycle. The # vertices simply stop these signals, keeping the $\cdot$ mark on their edges. When such a signal makes a complete tour, it comes back to the sender through a consecutive edge, and the sending vertex "knows" that there is an active face (see figure 3).

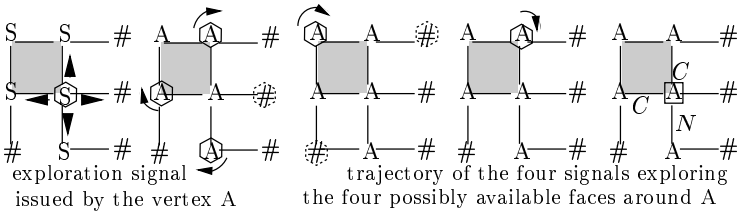

Figure3. $k = 4$ initialization steps.

After the exploration, if all signals which were sent by a vertex come back to it, then it changes to passive state.
Otherwise, it changes to active state, and

- for the sequences of circularly consecutive detected faces, it marks C on the first and the last edge, and $\cdot$ for the others.
- for the edges which do not belong to faces (neither left- nor rightwards), it marks N in the table.

The following rules apply to active vertices.

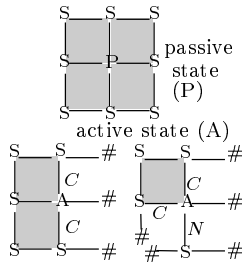

Figure4. Some examples for the start rule, for the central vertex

The algorithm

As we see, the algorithm uses a sequence of subgraphs of the initial graph G. We call active vertices and note with A the vertices of the current subgraph, for a given iteration. The vertices which have been active are noted with \$, and the other vertices, waiting to take part in the algorithm, are called passive and noted with P. There is also a leaf state noted L, and two leader state, noted B, and $\mathbb{B}$.
The active vertices are linked by edges of two types:

- the ones belonging to cycles (at start time being the borders of the components), noted with C, and
- the ones linking these components (through vertices), noted by N(see figure 6).

An iteration step occurs when the C-linked active vertices "wake up" dual vertices, towards the zone of passive vertices. The vertices which are (also) linked by N edges stay active, and new mixed edges can appear, linking a vertex to its dual vertex (see figure 6).

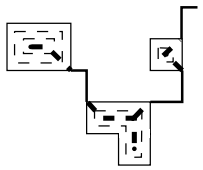

Figure5. Algorithm building the skeleton tree.

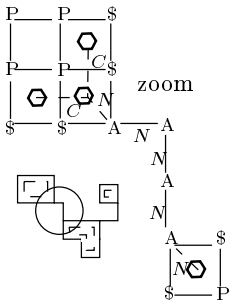

Figure6. Labels of one region of the graph.

Each vertex of $G + G^*$ has a table of $2d$ (respectively $2k$) elements of values from $\{C, N, \cdot\}$. The symbol $\cdot$ means the edge is not active, and will not be drawn in the figures. The cardinality of the table is due to the possible presence of mixed edges between a vertex and its dual, in addition to the normal d edges.

The shrinking process

The active vertices without N-marked edges and having exactly two C-marked edges change to \$ state. The dual vertices of faces having at least one active vertex become active, and update their tables by copying the corresponding values from the active vertices which "woke" them "up" (see figure 7).

If, among the vertices "waking up" the dual ones, there are some which remain active (because the have N-marked edges, and/or more than two C-marked edges), then mixed edges are created between these active vertices and the duals "woken up", and marked with N.

C to N switch — cycle splitting This rule slightly modifies the previous one : whenever a dual edge (x^*, y^*) has to be C-marked because two vertices x and y "woke up" x^* and y^*, and the edge (x, y) crosses the dual edge (x^*, y^*), then this last one is marked with N. The labeling allows the local detection of the crossing, because the label of (x, y) is of value between the values of the labels of (x, x^*) and (x, y^*). The same also holds for y. This rule locally stops the shrinking of a cycle, splitting it in two (or more) cycles, linked by elementary chain(s) (see figure 8).

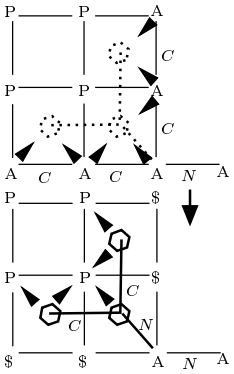

Figure7. An example for the dual switch

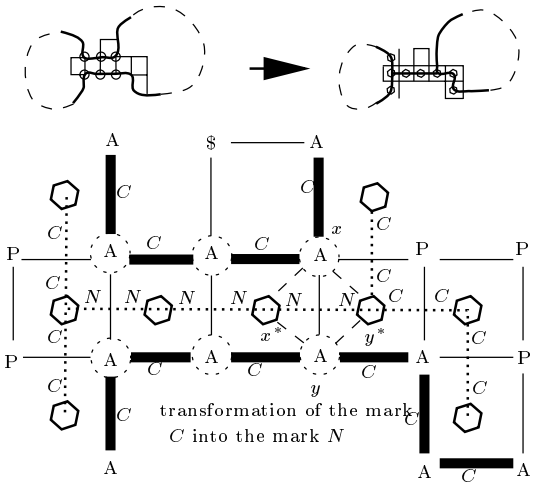

Figure8. An example for the shrinking.

Leafs If a vertex has only one active neighbor, and all the other neighbors in # or \$ state, it changes to L state. From this state, at the next step, if no neighbor is in L state, then the vertex changes to \$ state.

Election If an L-marked vertex has an L-marked neighbor, then both change to $\mathbb{B}$ and become leaders. If an active vertex has all its neighbors marked with L, then it changes to B, becoming unique leader. Also, if a vertex is "woken up" by a face, of k active vertices linked with C-marked edges, then it changes to B, becoming unique leader.

4 Analysis

In order to prove the correctness of the algorithm, we need to show certain properties of the subgraph induced by the active vertices, which is evolving in time.

We thus consider the sequence of subgraphs $G_t = (X_t, E_t)$ where $t \in \mathbb{N}$ represents time, with $X_t \subset X$ and $E_t \subset E$. X_t is defined as the set of all active vertices at time t of A_e, and E_t is the set of C, respectively N-marked edges of A_e at time t. Of course, $E_t \subset X_t \times X_t$. By cycle, from now on, we understand an elementary cycle.

Proposition 1. *At any moment t:*

- *any cycle of G_t is only made of C-marked edges, and any C-marked edge belongs to one and only one cycle of G_t.*
- *all passive vertices of A_e are inside the cycles of G_t, and these ones only contain passive vertices*
- *all \$- and #-marked vertices of A_e are outside the cycles of G_t.*
- *G_t is a connected graph.*

Remark 2. The N-marked edges of G_t belong to no cycle, but they link them. This induces a structure of "tree of cycles", as suggested in the presentation of this algorithm.

Proposition 2. *If the total number of passive vertices of A_e is strictly positive at time t, then it is also strictly greater than the total number of passive vertices of A_e at time $t+1$.*

If G_t is a tree with more than two vertices, then G_{t+1} has strictly less vertices than G_t.

It is very easy to prove these statements by induction, by making use of the local rules exposed in the previous section. Since at the beginning there is a finite number of passive vertices, and since and each iteration a non zero number of them irreversibly change to \$ state, G_t finally becomes a tree, which retracts itself until it has one or two vertices.

This gives us the temporal complexity as well, bounded by a sum of two terms. The first term is given by the shrinking of the cycles, which lasts for at most half of the longest of the diameters of the cycles of G_0. We call here *diameter of a cycle* the longest of the shortest paths *in* A_e from a vertex of a cycle to another vertex of the cycle.

The second term is the length of the longest path of a tree G_t, length which is bounded by the diameter of the initial graph A_e which contains it. Here the diameter of the graph is defined as the longest of the shortest paths in A_e from a vertex of the graph to another one.

Therefore, the algorithm elects a leader in $O(w)$, where w is the diameter of the graph A_e. This complexity is the same for the initial model, because the distance $w'(x, y)$ between two vertices x and y in the initial graph bounds from above the one in the auxiliary model, namely $w(x, y)$. For k the number of vertices of each face, we have $w(x, y) \leq w'(x, y)k/2$, because any path in the initial graph can be shortcut by passing through the dual vertex of each face of which part of the contour is used by the considered path (at most $k/2$ edges of the face). We thus can state the theorem.

Theorem 1. *Let k and d two integers. There exists an algorithm for the leader election problem for the finite subgraphs class of $\Gamma[k, d]$ without holes, in time $O(w)$, where w is the diameter of the considered subgraph.*

4.1 Finalization and optimization of the algorithm

The proposed algorithm leads to different possible final configurations: the configuration where there is only one leader, which is the configuration we want to reach, or a configuration with several leaders (edge or face). We shortly explain here how to avoid multi-leader final configurations, when possible.

Some cases are trivial, like a two leaders final configuration with one vertex of the initial graph and one vertex of the dual graph. The automaton just has to choose the vertex of the initial graph. But there are some non trivial cases.

It is obvious that final configurations with several leaders cannot be avoided when there is a perfect symmetry in the graph. An axial symmetry can lead to a leader-edge configuration and a central symmetry can even lead to leader face configuration. But in every other cases, we can reach an "only one leader" configuration. We describe now the algorithm, and mention that in the case starting from a k-leader face, it is k times slower, to allow the information exchange.

We associate to each vertex v an element s_v of $\{0; 1\}^d$, where the i-th component equal to 0 means the i-th neighbor is #, according to the (homogeneous) labeling.

The principle of the algorithm is to build a depth first search tree for each leader, and to compare at every step of the search the two new vertices reached by each leader in its search, using the d-tuples defined above, and the lexicographical order.

We can prove, by induction on the neighborhood of the leaders, that if the search is finished without finding any difference, there is a true symmetry in the graph, with an axis or a center between the two leaders. This finalization needs at most a time equal to $|V|$

In the case where the final configuration is a k-leader face, the process is slowed down (as said before), and, when a leader becomes a soldier, it simply stops its search. At the end, several configuration can be reached : one leader only left (no symmetry), k' leaders left, where k is divided by k' (central symmetry), or two leaders left (axial symmetry or a possible central one if k is even).

5 No actual need of the dual

As we have announced, the dual G^* is only used here in order to alleviate the presentation: the algorithm performs very well without any dual "physically" present. We do not need any supplemental information linking the initial and dual labelings; we

show how to perform the election only using the initial vertices and their homogeneous labeling.

The idea is very simple : instead of relying on alternating from one graph to the dual, the "true" computation bypasses this alternation, doing in one computation step two of the previously presented steps. There are several cases, as seen in figure 9:

1. the "normal" case, when the faces on the border have only one active part, a sub-chain of the whole cycle making up the face. Then, we end up, (bypassing the activation of the "middle" dual) with exactly the complementary of the face being active, a sort of swap.
2. "special" cases, when (in the superposition setting) a dual vertex has to remain active one step after the waking-up (or when the C-to-N switch occurs). Here, in our new setting, the whole face surrounding it has to remain active, with edges marked with a new mark, called F, behaving synchronously.

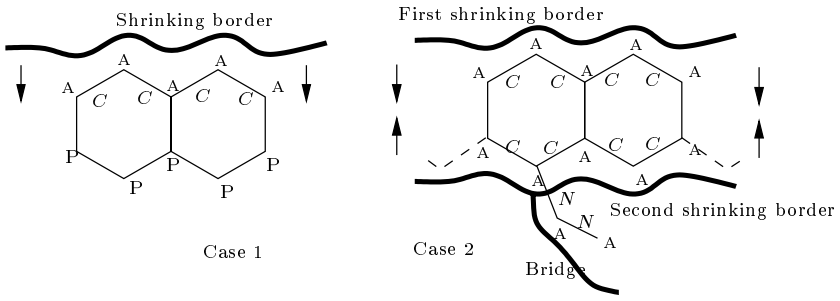

Figure9. "Normal" and "special" cases.

Care also has to be taken in order to preserve the connectivity with the N-marked incident edges. Thus an order of action has been chosen:

1. exploration of the face (through special signals sE) in order to detect the case (by active end-of-chain vertices)
2. deactivation and activation of the appropriate vertices (through other signals, detailed below)
3. edge label updates, if necessary, from C to N or to F.
4. local steps of the tree election (with synchronous decisions, when the edges are F-marked).

All these processes are performed in $O(k)$ time thus the overall complexity is not affected, since k is constant.

In the "normal" case, when only p vertices forming a chain are active among the k ones of a face, being also linked with C-marked edges. Both ends of the p-chain send exploration signals sE on all non C-marked edges and leading to P vertices. This means that sometimes, the signals will go round at the same time on many faces to which such an "end" of a p-chain belongs. The signals thus have to come back after going round each such face, and we know how to do this, using the homogeneous labeling. Now, for each face, the sender counts until sE comes to it; all the other vertices make it simply pass (and cross, when necessary, with an opposite sE) round

the face. This process is actually similar to the initialization process described in section 3.2, and shown in figure 3.

For each face, two cases appear:

- the signal sE comes back exactly after k tops. Then, the same vertex sends a new signal sA on that face. This signal follows the same trajectory as the sE, but also switches to A all the passive vertices. When reentering the p-chain (at its other end), it changes itself into another signal, s\$, which switches to \$ all the active vertices, deletes the C-marks, and it is finally stopped by the sender.
  If an external N-marked (or F-marked) edge is present (that is, adjacent to this face), then that vertex switches the signal s\$ to sN, which leaves active the next vertices, but adds to the C-marked edges the mark F. Of course, this signal is also stopped by the initial p-chain end (which sent the sE and the sA). Also, when two sA signals go along the same vertices because coming from two adjacent faces, then the C-mark is changed to the F-mark (this is the C-to-N switch).
- the signal sE comes back earlier. This means that the sE received is not the one physically sent by the same vertex, and we actually are in a more general case, when more chains of active vertices are on the same face. This is the tricky case (see figure 10), and we have to keep the whole face active (taking also care of any N-links). Now, the sender emits another signal, sF, which switches to active all the passive vertices, and to F all the edges, passing through all vertices. Thus even the C-marked edges are switched to F. Nothing happens when passing through vertices also linked with N, these links simply remaining active as they were.

Finally, we want that in this last case, the F-links behave like the N-ones, but synchronously. This is easy to achieve, by dilatating the time with a factor k : all "decisions" for the tree election shown in figure 2 are delayed until special interrogation signals go round the whole F-linked face, so that all vertices know what to do , and this takes the k supplemental steps.

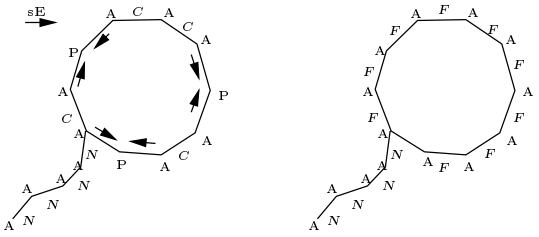

Figure10. Example of "special" case signals.

6 possible extensions

When not all the faces have the same number of vertices, it is easy to consider a union of all the transition functions locally defined for each type of face, and to apply the correct one at each vertex. The only conditions are that there be a bound on this number of vertices of the faces, and that the vertices know the exact cardinals of the faces they belong to.

The proofs can easily be extended to this case, because the notion of duality is still the same.

The extension to $\mathbb{Z}^n$ for $n > 2$ is more complicated, because due to the fact that some vertices can be in # state, locally, faces can be of less than n dimensions. In $\mathbb{Z}^n$, a face is a hypercube of dimension $p \leq n$, and the vertices need to detect the value p before starting the shrinking process.

This is easily done by the same mechanism of initial exploring of the neighborhood, and then, the appropriate dual vertices have to be simulated, that is dual of the appropriate p-dimensional faces. These in turn need to keep the mixed edges active, and also keep track of the limited dimension.

Globally the process is identical, and again, the proofs should easily be extended to this case as well.

A first question is how to extend this to graphs with "holes". There also is the open question of minimal time complexity needed to elect a leader. For instance, for $\mathbb{Z}^n$ graphs, in [7] and [5] a quadratic algorithm is given, in $O(w^2)$ time, where w is the diameter (the longest among the shortest paths) of the graph. Little+ is known about the possibility of improving this, or about an optimal bound. There is of course the trivial bound of linear time in the number of vertices, since all vertices need to at least "acknowledge" the achievement of the election.

References

1. A. Beckers and T. Worsch. A Perimeter-time CA for the Queen Bee Problem. In *Cellular Automata: Research towards Industry. (Proceedings of ACRI'98)*. Springer Verlag London, 1998.
2. H. S. M. Coxeter and W. O. J. Moser. *Generators and Relations for Discrete Groups*, volume 14 of *Ergebnisse der Mathematik und ihrer Grenzengebiete - neue Folge*. Springer Verlag, 1965.
3. J. Mazoyer and M. Delorme, editors. *Cellular Automata, a Parallel Model*, chapter An Introduction to Automata on Graphs, by E. Rémila, pages 345–352. Kluwer Academic Publishers, 1999.
4. J. Mazoyer, C. Nichitiu, and E. Remila. Compass Permits Leader Election. In *Proceedings of SODA 1999*. SIAM, 1999.
5. C. Nichitiu. *Algorithmique sur graphes d'automates : élection d'un chef, simulations*. PhD thesis, École Normale Supérieure de Lyon, http://www.ens-lyon.fr, December 1999.
6. C. Nichitiu and E. Rémila. Simulations of graph automata. In Thomas Worsch and Roland Vollmar, editors, *MFCS'98 Satellite Workshop on Cellular Automata*. Universität Karlsruhe, Interner Bericht 19/98, 1998.
7. C. Nichitiu and E. Rémila. Leader Election by d Dimensional Cellular Automata. In *Proceedings of ICALP 1999*, volume LNCS 1644. Springer Verlag, 1999.
8. A. Wu and A. Rosenfeld. Cellular Graph Automata. I. Basic Concepts, Graph Property Measurement, Closure Properties. *Information and Control*, 42:305–329, 1979.