Election, Naming and Cellular Edge Local Computations

Jérémie Chalopin, Yves Métivier, Wieslaw Zielonka

Election, Naming and Cellular Edge Local Computations (Extended Abstract)

Jérémie Chalopin[1], Yves Métivier[1], and Wiesław Zielonka[2]

[1] LaBRI, Université Bordeaux I, ENSEIRB,
351 cours de la Libération, 33405 Talence, France
{chalopin,metivier}@labri.fr
[2] LIAFA, Université Paris 7
2, place Jussieu, case 7014, 75251 Paris Cedex 05, France
zielonka@liafa.jussieu.fr

Abstract. We examine the power and limitations of the weakest vertex relabelling system which allows to change a label of a vertex in function of its own label and of the label of one of its neighbours. We characterise the graphs for which two important distributed algorithmic problems are solvable in this model: naming and election.

1 Introduction

The role of the local computation mechanisms is fundamental for delimiting the borderline between positive and negative results in distributed computation. Understanding the power of local computations in different models enhances our understanding of basic distributed algorithms. Yamashita and Kameda [11], Boldi and al. [3], Mazurkiewicz [8] and Chalopin and Métivier [4] characterise families of graphs in which election is possible under different models of distributed computations. Even if these results cover a broad class of models there are still a few natural models which were not yet examined. We consider here one of such models where an elementary computation step modifies the state of one network vertex and this modification depends on its current state and on the state of one of its neighbours. We solve, in this model, two important algorithmic problems: the election problem and the naming problem, which turn out to be not equivalent. We give the characterisation of graphs which admit distributed solutions for both problems in this model.

To this end we find suitable graph morphisms that enable to formulate conveniently the necessary conditions in the spirit of Angluin [1]. It turns out that in our case the relevant morphisms are graph submersions. The presented conditions are also sufficient: algorithms, inspired by Mazurkiewicz [8], are given, that enable to solve the naming and the election problems for corresponding graphs.

1.1 Our Model

A network of processors will be represented as a connected undirected graph $G = (V(G), E(G))$ without self-loop and multiple edges. As usual the vertices represent processors and edges direct communication links. The state of each processor is represented by the label $\lambda(v)$ of the corresponding vertex. An elementary computation step will be represented by relabelling rules of the form given schematically in Figure 1. The computations using uniquely this type of relabelling rules are called in our paper *cellular edge local computations*. Thus an algorithm in our model is simply given by some (possibly infinite but always recursive) set of rules of the type presented in Figure 1. A run of the algorithm consists in applying the relabelling rules specified by the algorithm until no rule is applicable, which terminates the execution. The relabelling rules are applied asynchronously and in any order, which means that given the initial labelling usually many different runs are possible.

Fig. 1. Graphical form of a rule for cellular edge local computations. If in a graph G there is a vertex labelled X with a neighbour labelled Y then applying this rule we replace X by a new label X'. The labels of all other graph vertices are irrelevant for such a computation step and remain unchanged. The vertex of G changing the label will be called *active* and filled with black, the neighbour vertex used to match the rule is called *passive* and marked as unfilled on the figure. All the other vertices of G not participating in such elementary relabelling step are called *idle*.

1.2 Election, Naming and Enumeration

The election problem is one of the paradigms of the theory of distributed computing. It was first posed by LeLann [6]. A distributed algorithm solves the election problem if it always terminates and in the final configuration exactly one processor is marked as *elected* and all the other processors are *non elected*. Moreover, it is supposed that once a processor becomes *elected* or *non elected* then it remains in such a state until the end of the algorithm. Elections constitute a building block of many other distributed algorithms since the elected vertex can be subsequently used to make some centralised decisions, to initialise some other activity, to centralise or to broadcast information etc.

The generic conditions listed above, required for an election algorithm, have a direct translation in our model: we are looking for a relabelling system where each run terminates with exactly one vertex labelled *elected* and all the other vertices labelled as *non elected*. Again we require that no rule allows to change either an *elected* or a *non-elected* label.

The naming problem is another important problem in the theory of distributed computing. The aim of a naming algorithm is to arrive at a final config-

uration where all processors have unique identities. To be able to give dynamically and in a distributed way unique identities to all processors is very important since many distributed algorithm work correctly only under the assumption that all processors can be unambiguously identified.

The enumeration problem is a variant of the naming problem. The aim of a distributed enumeration algorithm is to attribute to each network vertex a unique integer in such a way that this yields a bijection between the set $V(G)$ of vertices and $\{1, 2, \ldots, |V(G)|\}$.

We also distinguish two kinds of termination: the implicit one that simply means that the algorithm always terminates, and the explicit one that means that at least one node can detect that the algorithm has terminated. Obviously, if we can solve the naming problem with an explicit termination then we can also elect, for example the vertex with the smallest or the greatest identity.

The naming and the election problems are often equivalent for various computational models [8, 4], however this is not the case for our model. It turns out that in our model the class of graphs for which naming is solvable admits a simple and elegant characterisation; unfortunately a similar characterisation for the election problem is quite involved.

1.3 Overview of our Results

Under the model of cellular edge local computations, we present a complete characterisation of graphs for which naming and election are possible: Theorems 7 and 11. The problems are solved constructively, we present naming and election algorithms that work correctly for all graphs where these problems are solvable. Imposed space limitations do not allow to present the correctness proofs for our algorithms.

1.4 Related Works

The election problem was already studied in a great variety of models [2, 7, 10]. The proposed algorithms depend on the type of the basic computation steps, they work correctly only for a particular type of a network topology (tree, grid, torus, ring with a known prime number of vertices etc.) or it is assumed that some initial extra knowledge is available to processors.

Yamashita and Kameda [11] consider the model where, in each step, one of the vertices, depending on its current label, either changes the label, or sends/receives a message via one of its ports. They proved that there exists an election algorithm for G if and only if the symmetricity of G is equal to 1, where the symmetricity depends on the number of labelled trees isomorphic to a certain tree associated with G ([11], Theorem 1 p. 75).

Mazurkiewicz [8] considers the asynchronous computation model presented in Figure 2. His characterisation of the graphs where enumeration/election are possible is based on the notion of non ambiguous graphs and may be formulated equivalently using coverings [5]. He gives a nice and simple enumeration algorithm for the graphs minimal for the covering relation.

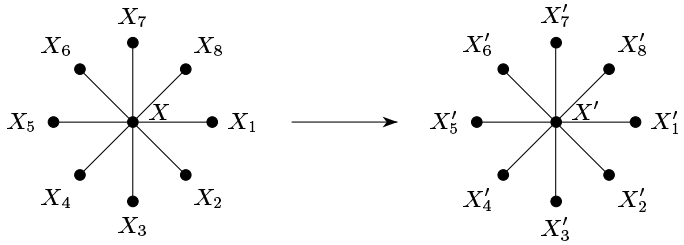


Fig. 2. In the model of Mazurkiewicz [8] a chosen vertex can change its label together with all its neighbours. The relabelling rules have therefore the form presented on this figure. Note that this involves a much greater degree of synchronisation than in the systems that we examine in our paper.

Boldi and al. [3] consider a model where the network is a directed multigraph G and contrary to our model they allow also arc labellings. When a processor is activated, it changes its state depending on its previous state and on the states of its ingoing neighbours; the outgoing neighbours do not participate in such an elementary computation step. They investigate two modes of computation: synchronous and asynchronous while in our paper only asynchronous computations are examined. In their study, they use fibrations which are generalisations of coverings. Boldi and al. [3] prove that there exists an election algorithm in their model for a graph G if and only if G is not properly fibred over another graph H (for the asynchronous case, they only consider discrete fibrations). To obtain this characterisation, they use the same mechanism as Yamashita and Kameda: each node computes its own view and next the node with the weakest view is elected.

In [4], three different asynchronous models are examined. Schematically, the rules of all three models are presented in Figure 3. Note that, contrary to the model we examine in the present paper, all these models allow edge labelling. It turns out that for all models described in Figure 3 naming and election are equivalent. In [4], it is proved that for all models described in Figure 3 the election and naming problems can be solved on a graph G if and only if G is not a covering of any graph H not isomorphic to G, where H can have multiple edges but no self-loop.

We can note that, although the model studied in this paper and model A in Figure 3 seem to be very close, the characterisations of graphs for which the naming problem and the election problem can be solved in these models are very different. The intuitive reason is that if we allow to label the edges then each processor can subsequently consistently identify the neighbours. On the other hand, in the model that we examine here, since edges are no more labelled, a vertex can never know if it synchronises with the same neighbour or another one.

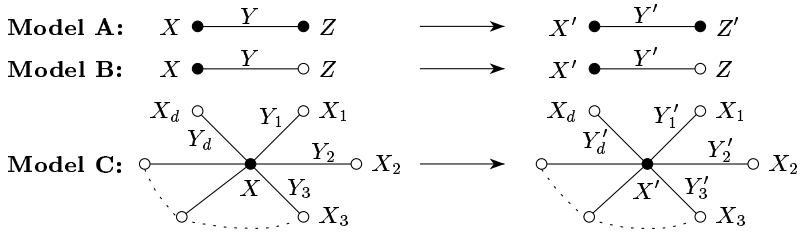


Fig. 3. Elementary relabelling steps for the three models examined in [4].

2 Preliminaries

We consider finite, undirected, connected graphs $G = (V(G), E(G))$ with vertices $V(G)$ and edges $E(G)$ without multiple edges or self-loop. Two vertices u and v are said to be adjacent or neighbours if $\{u, v\}$ is an edge of G (thus u and v are necessarily distinct since no self-loop is admitted) and $N_G(v)$ will stand for the set of neighbours of v. An edge e is incident to a vertex v if $v \in e$ and $I_G(v)$ will stand for the set of all the edges of G incident to v. The degree of a vertex v, denoted $d_G(v)$, is the number of edges incident with v.

A homomorphism between graphs G and H is a mapping $\gamma\colon V(G) \to V(H)$ such that if $\{u, v\} \in E(G)$ then $\{\gamma(u), \gamma(v)\} \in E(H)$. Since our graphs do not have self-loop, this implies that $\gamma(u) \neq \gamma(v)$ whenever u and v are adjacent.

We say that γ is an isomorphism if γ is bijective and γ^{-1} is a homomorphism. A class of graphs will be any set of graphs containing all graphs isomorphic to some of its elements. A graph H is a subgraph of G, noted $H \subseteq G$, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. An occurrence of H in G is an isomorphism γ between H and a subgraph H' of G.

For any set S, $|S|$ denotes the cardinality of S while $\mathcal{P}_{\text{fin}}(S)$ is the set of finite subsets of S. For any integer q, we denote by $[1, q]$ the set of integers $\{1, 2, \ldots, q\}$.

Throughout the paper we will consider graphs where vertices are labelled with labels from a recursive label set L. A graph labelled over L is a couple $\mathbf{G} = (G, \lambda)$, where G is an underlying non labelled graph and $\lambda\colon V(G) \to L$ is a (vertex) labelling function. The class of graphs labelled by L will be denoted by $\mathcal{G}_L$.

Let H be a subgraph of G and λ_H the restriction of a labelling $\lambda\colon V(G) \to L$ to $V(H)$. Then the labelled graph $\mathbf{H} = (H, \lambda_H)$ is called a subgraph of $\mathbf{G} = (G, \lambda)$; we note this fact by $\mathbf{H} \subseteq \mathbf{G}$. A homomorphism of labelled graphs is just a labelling-preserving homomorphism of underlying unlabelled graphs.

Submersions are locally surjective graph morphisms:

Definition 1. *A graph G is a submersion of a graph H via a morphism $\gamma\colon G \to H$ if $\forall v \in V(G)$, γ is surjective on the neighbourhood $N_G(v)$, that is $\gamma(N_G(v)) = N_H(\gamma(v))$. The graph G is a* proper submersion *of H if γ is not an isomorphism; G is* submersion-minimal *if G is not a proper submersion of any other graph.*

Naturally, submersions of labelled graphs are just submersions of underlying unlabelled graphs preserving the labelling.

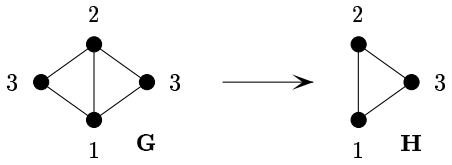


Fig. 4. The labelled graph $\mathbf{G}$ is a submersion of $\mathbf{H}$ via the mapping γ which maps each vertex of $\mathbf{G}$ labelled i to the unique vertex of $\mathbf{H}$ with the same label i. This submersion is proper and the graph $\mathbf{H}$ is itself submersion-minimal.

For any set $\mathcal{R}$ of edge local relabelling rules of the type described in Figure 1 we shall write $\mathbf{G}\ \mathcal{R}\ \mathbf{G}'$ if $\mathbf{G}'$ can be obtained from $\mathbf{G}$ by applying a rule of $\mathcal{R}$ on some edge of $\mathbf{G}$. Obviously, $\mathbf{G}$ and $\mathbf{G}'$ have the same underlying graph G, only the labelling changes for exactly one (active) vertex. Thus, slightly abusing the notation, $\mathcal{R}$ will stand both for a set of rules and the induced relabelling relation over labelled graphs. The transitive closure of such a relabelling relation is noted $\mathcal{R}^*$.

The relation $\mathcal{R}$ is called *noetherian* on a graph $\mathbf{G}$ if there is no infinite relabelling sequence $\mathbf{G}_0\ \mathcal{R}\ \mathbf{G}_1\ \mathcal{R}\ \ldots$, with $\mathbf{G}_0 = \mathbf{G}$. The relation $\mathcal{R}$ is noetherian on a set of graphs if it is noetherian on each graph of the set. Finally, the relation $\mathcal{R}$ is called noetherian if it is noetherian on each graph.

Clearly noetherian relations code always terminating algorithms.

The following simple observation exhibits a strong link between submersions and cellular edge local relabellings. This is a counterpart of the lifting lemma of Angluin [1] adapted to submersions.

Lemma 2 (Lifting Lemma). *Let $\mathcal{R}$ be a cellular edge locally generated relabelling relation and let $\mathbf{G}$ be a submersion of $\mathbf{H}$. If $\mathbf{H}\ \mathcal{R}^*\ \mathbf{H}'$ then there exists $\mathbf{G}'$ such that $\mathbf{G}\ \mathcal{R}^*\ \mathbf{G}'$ and $\mathbf{G}'$ is a submersion of $\mathbf{H}'$.*

Proof. It is sufficient to prove the lemma for one step of the relabelling. Let $\varphi : \mathbf{G} \to \mathbf{H}$ be a submersion, $\mathbf{G} = (G, \lambda), \mathbf{H} = (H, \nu)$. Suppose that a cellular edge rule is applied to an active vertex $v \in V(H)$ yielding a new labelling ν' on H. Then, since φ is a submersion, all vertices of $\varphi^{-1}(v)$ are pairwise non adjacent and therefore we can apply the same relabelling rule to all vertices of $\varphi^{-1}(v)$ in G, in any order. This yields a labelling λ' on G such that $\varphi : (G, \lambda') \to (H, \nu')$ remains a submersion. Note that we have simulated here one step relabelling in $\mathbf{H}$ by several relabellings in $\mathbf{G}$ that use the same rule. □

3 Enumeration and Naming Problems

We prove that there exists no naming algorithm and no enumeration algorithm on a graph $\mathbf{G}$ using cellular edge local computations if the graph is not submersion-minimal. The proof is analogous to that of Angluin [1].

Proposition 3. *Let* $\mathbf{G}$ *be a labelled graph which is not submersion-minimal. There is no naming algorithm for* $\mathbf{G}$ *and no enumeration algorithm for* $\mathbf{G}$ *using cellular edge local computations.*

Proof. Let $\mathbf{H}$ be a labelled graph not isomorphic to $\mathbf{G}$ such that $\mathbf{G}$ is a submersion of $\mathbf{H}$ via φ. For every cellular edge local algorithm $\mathcal{R}$, consider an execution of $\mathcal{R}$ on $\mathbf{H}$ that leads to a final configuration $\mathbf{H}'$. From Lemma 2, there exists an execution of $\mathcal{R}$ on $\mathbf{G}$ such that the final configuration $\mathbf{G}' = (G, \lambda')$ is a submersion of $\mathbf{H}'$. Since $\mathbf{G}'$ is not isomorphic to $\mathbf{H}'$, there exist distinct $v, v' \in V(G)$ such that $\lambda'(v) = \lambda'(v')$. Consequently, $\mathcal{R}$ does not solve either the naming or the enumeration problem on $\mathbf{G}$. □

3.1 An Enumeration Algorithm

In this section, we describe a Mazurkiewicz-like algorithm $\mathcal{M}$ using cellular edge local computations that solves the enumeration problem on a submersion-minimal graph $\mathbf{G}$.

Each vertex v attempts to get its own number between 1 and $|V(G)|$. A vertex chooses a number and exchanges its number with its neighbours. If a vertex u discovers the existence of another vertex v with the same number, then it compares its *local view* (the numbers of its neighbours) with the local view of v. If the label of u or the local view of u is "weaker", then u chooses another number and broadcasts it again with its local view. At the end of the computation, every vertex will have a unique number if the graph is submersion-minimal.

We consider a graph $\mathbf{G} = (G, \lambda)$ with an initial labelling $\lambda\colon V(G) \to L$. During the computation each vertex $v \in V(G)$ will acquire new labels of the form $(\lambda(v), n(v), N(v), M(v))$, where:

- the first component $\lambda(v)$ is just the initial label (and thus remains fixed during the computation),
- $n(v) \in \mathbb{N}$ is the current *identity number* of v computed by the algorithm,
- $N(v) \in \mathcal{P}_{\text{fin}}(\mathbb{N})$ is the *local view* of v. Intuitively, the algorithm will try to update the current view in such a way that $N(v)$ will consist of current identities of the neighbours of v. Therefore $N(v)$ will be always a finite (possibly empty) set of integers,
- $M(v) \subseteq \mathbb{N} \times L \times \mathcal{P}_{\text{fin}}(\mathbb{N})$ is the current *mailbox* of v. It contains the whole information received by v during the computation.

The fundamental property of the algorithm is based on a total order on the set $\mathcal{P}_{\text{fin}}(\mathbb{N})$ of local views, as defined by Mazurkiewicz [8].

Let $N_1, N_2 \in \mathcal{P}_{\text{fin}}(\mathbb{N})$, $N_1 \neq N_2$. Then $N_1 \prec N_2$ if the maximal element of the symmetric difference $N_1 \triangle N_2 = (N_1 \setminus N_2) \cup (N_2 \setminus N_1)$ belongs to N_2. Note that in particular the empty set is minimal for $\prec$.

It can be helpful to note that the order $\prec$ is just a reincarnation of the usual lexicographic order. Let $n_1, n_2, \ldots, n_k$ and $m_1, m_2, \ldots, m_l$ be all elements of N_1 and N_2 respectively listed in the decreasing order (decreasing for the usual order over integers): $n_1 > n_2 > \cdots > n_k$ and $m_1 > m_2 > \cdots > m_l$. Then $N_1 \prec N_2$ iff

either *(i)* $k \leq l$ and for all i, $1 \leq i \leq k$, $n_i = m_i$ or *(ii)* $n_i < m_i$ where i is the smallest index such that $n_i \neq m_i$.

If $N(u) \prec N(v)$ then we say that the local view $N(v)$ of v is stronger than the one of u (and $N(u)$ is weaker than $N(v)$).

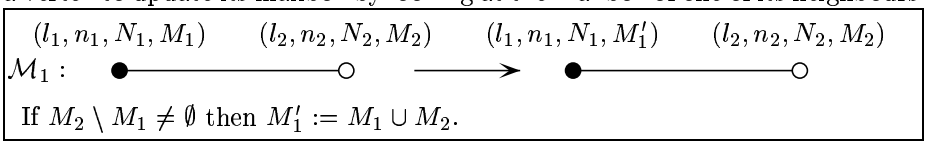
We assume for the rest of this paper that the set of labels L is totally ordered by $<_L$.

Finally, we extend $\prec$ to a total order on $L \times \mathcal{P}_{\text{fin}}(\mathbb{N})$: $(l, N) \prec (l', N')$ if either $l <_L l'$ or ($l = l'$ and $N \prec N'$).

Occasionally we shall use the reflexive closure $\preceq$ of $\prec$.

We describe here the relabelling rules that define the enumeration algorithm.

First of all, to launch the algorithm there is a special initial rule $\mathcal{M}_0$ that just extends the initial label $\lambda(v)$ of each vertex v to $(\lambda(v), 0, \emptyset, \emptyset)$. The rules $\mathcal{M}_1$ and $\mathcal{M}_2$ are close to the rules used by Mazurkiewicz [8]. The first rule $\mathcal{M}_1$ enables a vertex to update its mailbox by looking at the mailbox of one of its neighbours:



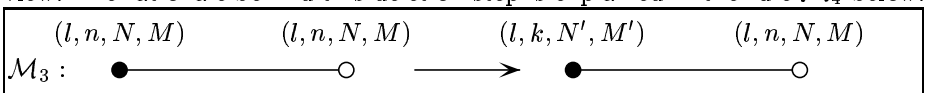
If $M_2 \setminus M_1 \neq \emptyset$ then $M_1' := M_1 \cup M_2$.

The second rule $\mathcal{M}_2$ does not involve any synchronisation with a neighbour vertex. It enables a vertex v to change its identity if the current identity number $n(v)$ is 0 or if the mailbox of v contains a message from a vertex with the same identity but with a stronger label or a stronger local view.

If $n = 0$ or there exists $(n, \ell', N') \in M$ such that $(\ell, N) \prec (\ell', N')$ then $k := 1 + \max\{n' \mid \exists (n', \ell', N') \in M\}$ and $M' := M \cup \{(k, \ell, N)\}$.

(In the formula above we assume that max of an empty set is 0.)

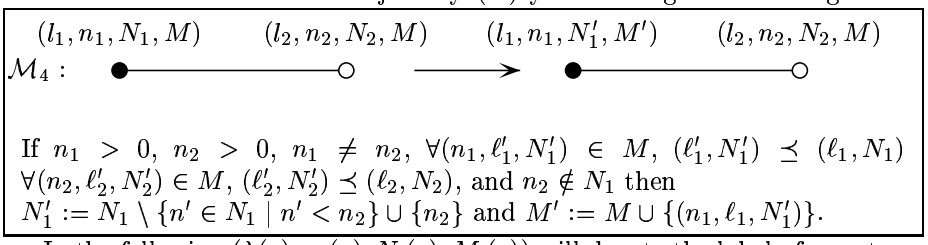
The third rule $\mathcal{M}_3$ allows to change the current identity for a vertex v having a neighbour v' with exactly the same current label (all four components should be identical). Moreover, at the same step, the identity $n(v')$ of the neighbour v' of v is inserted into the local view $N(v)$ and at the same time all the elements m of $N(v)$ such that $m < n(v')$ are deleted from the local view. The rationale behind this deletion step is explained in the rule $\mathcal{M}_4$ below.



If $n > 0$ and $\forall (n, \ell', N') \in M$, $(\ell', N') \preceq (\ell, N)$ then $k := 1 + \max\{n' \mid \exists (n', \ell', N') \in M\}$, $N' := N \setminus \{m \in N \mid m < n\} \cup \{n\}$ and $M' := M \cup \{(k, \ell, N')\}$.

The fourth rule $\mathcal{M}_4$ enables a vertex v to add the current identity number $n(v')$ of one of its neighbours to its local view $N(v)$. As for the preceding rule, all the elements m belonging to $N(v)$ such that $m < n(v')$ are deleted from the current view.

The intuitive justification for the deletion of all such m is the following. Let us suppose that the vertex v synchronises with a neighbour v' and observes that the current identity number $n(v')$ of v' does not belong to his current view $N(v)$. Then, since the very purpose of the view $N(v)$ is to stock the identity numbers of all the neighbours, we should add $n(v')$ to the view $N(v)$ of v. But now two cases arise. If v synchronises with v' for the first time then adding $n(v')$ to the view of v is sufficient. However, it can also be the case that v synchronised with v' in the past and in the meantime v' has changed its identity number. Then v should not only add the new identity number $n(v')$ to its view but, to remain in a consistent state, we should delete the old identity number of v' from the local view of v. The trouble is that v has no means to know which of the numbers present in its view $N(v)$ should be deleted and it is even unable to decide which of the two cases holds (first synchronisation with v' or not). However, since our algorithm assures the monotonicity of subsequent identity numbers of each vertex, we know that the eventual old identity number of v' is less than the current identity $n(v')$. Therefore, by deleting all $m < n(v')$ from the local view $N(v)$ we are sure to delete all invalid information. Of course, in this way we risk to delete also the legitimate current identities of other neighbours of v from its view $N(v)$. However, this is not a problem since v can recover this information just by (re)synchronising all such neighbours.



If $n_1 > 0$, $n_2 > 0$, $n_1 \neq n_2$, $\forall (n_1, \ell_1', N_1') \in M$, $(\ell_1', N_1') \preceq (\ell_1, N_1)$ $\forall (n_2, \ell_2', N_2') \in M$, $(\ell_2', N_2') \preceq (\ell_2, N_2)$, and $n_2 \notin N_1$ then
$N_1' := N_1 \setminus \{n' \in N_1 \mid n' < n_2\} \cup \{n_2\}$ and $M' := M \cup \{(n_1, \ell_1, N_1')\}$.

In the following $(\lambda(v), n_i(v), N_i(v), M_i(v))$ will denote the label of a vertex v after the ith computation step of the algorithm $\mathcal{M}$ given above.

The algorithm has some remarkable monotonicity properties:

Lemma 4. *For each step i and each vertex v: (A) $n_i(v) \leq n_{i+1}(v)$, (B) $N_i(v) \preceq N_{i+1}(v)$, and (C) $M_i(v) \subseteq M_{i+1}(v)$. Moreover, there exists at least one vertex v such that at least one of these inequalities/inclusions is strict for v.*

The local knowledge of a vertex v reflects to some extent some real properties of the current configuration:

Lemma 5. *Let $v \in V(G)$. If $(m, \ell, N) \in M_i(v)$ then for some vertex $w \in V(G)$, $n_i(w) = m$. If $n_i(v) \neq 0$ and $(m', \ell', N') \in M_i(v)$ then, for every $1 \leq m \leq m'$, there exist ℓ and N such that $(m, \ell, N) \in M_i(v)$.*

This fact allows to deduce the following properties of the final labelling:

Lemma 6. *Any run ρ of the enumeration algorithm on a connected labelled graph $\mathbf{G} = (G, \lambda)$ terminates and yields a final labelling $(\lambda, n_\rho, N_\rho, M_\rho)$ satisfying the following conditions:*

(1) Let m be the maximal number in the final labelling, $m = \max\{n_\rho(v) \mid v \in V(G)\}$. Then for every $1 \leq p \leq m$ there is some $v \in V(G)$ with $n_\rho(v) = p$,

and for all vertices v, v':

(2) $M_\rho(v) = M_\rho(v')$,
(3) $(\lambda(v), n_\rho(v), N_\rho(v)) \in M_\rho(v')$,
(4) $n_\rho(v) = n_\rho(v')$ implies that $\lambda(v) = \lambda(v')$ and $N(v) = N(v')$,
(5) $n \in N_\rho(v)$ if and only if there exists $w \in N_G(v)$ such that $n_\rho(w) = n$; in this case, $n_\rho(v) \in N_\rho(w)$.

Under the notation of Lemma 6 we can construct the labelled graph $\mathbf{H}_\rho$: the vertices of $\mathbf{H}_\rho$ are integers $n_\rho(v)$, i.e. final identity numbers, each $n_\rho(v)$ labelled by $\lambda(v)$ (this labelling is well defined by Lemma 6 (4)) and with edges naturally inherited from $\mathbf{G}$. In fact, the mapping n_ρ is a submersion from $\mathbf{G}$ to $\mathbf{H}_\rho$. This observation yields:

Theorem 7. *For every graph $\mathbf{G}$, the following statements are equivalent:*

(i) there exists a naming algorithm on $\mathbf{G}$ using cellular edge local computations,
(ii) there exists a naming algorithm with termination detection on $\mathbf{G}$ using cellular edge local computations,
(iii) there exists an enumeration algorithm on $\mathbf{G}$ using cellular edge local computations,
(iv) there exists an enumeration algorithm with termination detection on $\mathbf{G}$ using cellular edge local computations,
(v) the graph $\mathbf{G}$ is a submersion-minimal graph.

4 Election Problem

If we can solve the enumeration problem then we can solve the election problem; once a vertex gets the identity number $|V(G)|$ we declare it *elected*.

Nevertheless, in our model, the enumeration and the election problems are not equivalent. The graph $\mathbf{G}$ in Figure 5 is not submersion-minimal, since the morphism from $\mathbf{G}$ to $\mathbf{H}$ induced by the labelling of $\mathbf{G}$ is locally surjective and therefore neither the enumeration nor the naming problem can be solved on $\mathbf{G}$. But let us execute the preceding algorithm on $\mathbf{G}$. At the end, the vertex labelled 3 in $\mathbf{G}$ will know that it is unique with at least three different neighbours and therefore can declare itself as elected.

We would like to give here necessary conditions characterising the graphs with solvable election problem. Given a graph $\mathbf{G}$, we denote by $\mathcal{S}_\mathbf{G}$ the set of graphs $\mathbf{H}$ such that there exists a submersion from $\mathbf{G}$ onto $\mathbf{H}$. From Lemma 2, any algorithm $\mathcal{A}$ that solves the election problem on $\mathbf{G}$ using cellular edge local computations will solve the election problem on every graph $\mathbf{H} \in \mathcal{S}_\mathbf{G}$.

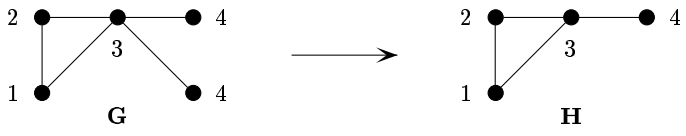


Fig. 5. A graph for which we can solve the election problem but not the enumeration problem.

Remark 8. Consider an algorithm $\mathcal{A}$ that solves the election problem on $\mathbf{G}$. Suppose that there exists a subgraph $\mathbf{G}'$ of $\mathbf{G}$ that is a submersion of a graph $\mathbf{H} \in \mathcal{S}_{\mathbf{G}}$ via a morphism φ. If there exists an execution of $\mathcal{A}$ on $\mathbf{H}$ that elects a vertex $v \in V(H)$ such that $|\varphi^{-1}(v)| > 1$, then there exists an execution of $\mathcal{A}$ on $\mathbf{G}'$ such that the label *elected* appears at least twice. Since each execution of $\mathcal{A}$ on $\mathbf{G}'$ can be extended to an execution of $\mathcal{A}$ on $\mathbf{G}$, there exists an execution of $\mathcal{A}$ over $\mathbf{G}$ that leads to the election of at least two vertices, this is in contradiction with the choice of $\mathcal{A}$. We can therefore define $P_{\mathbf{H}}(\mathbf{G}', \varphi) = \{v \in V(H) \mid |\varphi^{-1}(v)| > 1\}$ and each execution of $\mathcal{A}$ on $\mathbf{H}$ cannot elect a vertex $v \in P_{\mathbf{H}}(\mathbf{G}', \varphi)$.

Consider a graph $\mathbf{H} \in \mathcal{S}_{\mathbf{G}}$. Let $P_{\mathbf{H}}(\mathbf{G})$ be the union of all $P_{\mathbf{H}}(\mathbf{G}', \varphi)$ for φ ranging over all submersions of subgraphs $\mathbf{G}'$ of $\mathbf{G}$ to $\mathbf{H}$ and $C_{\mathbf{H}}(\mathbf{G}) = V(H) \setminus P_{\mathbf{H}}(\mathbf{G})$ (the elements of this set are called the candidates of $\mathbf{H}$ for $\mathbf{G}$). From Remark 8, every election algorithm $\mathcal{A}$ over $\mathbf{G}$ must be such that each execution of $\mathcal{A}$ over $\mathbf{H}$ should elect a vertex in $C_{\mathbf{H}}(\mathbf{G})$. Consequently, if there exists an election algorithm $\mathcal{A}$ on $\mathbf{G}$ then for every graph $\mathbf{H} \in \mathcal{S}_{\mathbf{G}}$, $C_{\mathbf{H}}(\mathbf{G}) \neq \emptyset$.

Suppose that there exist two disjoint subgraphs $\mathbf{G}_1$ and $\mathbf{G}_2$ of $\mathbf{G}$ such that $\mathbf{G}_1$ (resp. $\mathbf{G}_2$) is a submersion of a graph $\mathbf{H}_1 \in \mathcal{S}_{\mathbf{G}}$ (resp. $\mathbf{H}_2 \in \mathcal{S}_{\mathbf{G}}$). Then there does not exist any election algorithm using cellular edge local computations. Indeed, otherwise, there exists an execution of the algorithm on $\mathbf{G}$ such that the label *elected* appears once in $\mathbf{G}_1$ and once in $\mathbf{G}_2$, which is impossible for an election algorithm. Recapitulating:

Proposition 9. *Let $\mathbf{G}$ be a labelled graph such that there exists an election algorithm for $\mathbf{G}$ using cellular edge local computations. Then the following conditions are satisfied:*

1. *for every $\mathbf{H} \in \mathcal{S}_{\mathbf{G}}$, $C_{\mathbf{H}}(\mathbf{G}) \neq \emptyset$,*
2. *there do not exist two disjoint subgraphs $\mathbf{G}_1$ and $\mathbf{G}_2$ of $\mathbf{G}$ such that $\mathbf{G}_1$ (resp. $\mathbf{G}_2$) is a submersion of a graph $\mathbf{H}_1 \in \mathcal{S}_{\mathbf{G}}$ (resp. $\mathbf{H}_2 \in \mathcal{S}_{\mathbf{G}}$).*

4.1 An Election Algorithm

We now consider a graph $\mathbf{G}$ satisfying the conditions of Proposition 9.

Our aim is to present an algorithm such that each execution over $\mathbf{G}$ will detect a graph $\mathbf{H} \in \mathcal{S}_{\mathbf{G}}$ such that there exists a subgraph $\mathbf{G}'$ of $\mathbf{G}$ that is a submersion of $\mathbf{H}$.

To this end we adapt the enumeration algorithm from the preceding section and the termination detection algorithm of Szymansky, Shi and Prywes [9].

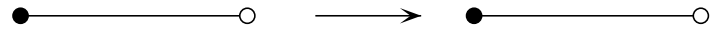
The idea is to execute the enumeration algorithm given for a graph and to reconstruct a graph from the mailboxes of the nodes. If the reconstructed graph is an element of $\mathcal{S}_{\mathbf{G}}$, the nodes check if they all agree on this graph.

As in Section 3.1, we start with a labelled graph $\mathbf{G} = (G, \lambda)$. During the computation vertices v will get new labels of the form $(\lambda(v), n(v), N(v), M(v), a(v), H(v))$ representing the following information (again the first component $\lambda(v)$ remains fixed) :

- $n(v) \in \mathbb{N}$ is the *identity number* of the vertex v computed by the algorithm,
- $a(v) \in \mathbb{N}$ is the confidence level of the vertex v,
- $N(v)$ is the *local view* of v. If the vertex v has a neighbour v', relabelling rules will allow v to add the couple $(n(v'), a(v'))$ to $N(v)$. Thus $N(v)$ is always a finite set of couples of integers. For $N \in \mathcal{P}_{\text{fin}}(\mathbb{N}^2)$, we note $\Pi_1(N) = \{n \mid \exists (n, a) \in N\}$ the projection on the first component.
- $M(v) \subseteq \mathbb{N} \times L \times \mathcal{P}_{\text{fin}}(\mathbb{N})$ is the *mailbox* of v and contains the information received by v about the identity numbers existing in the graph and the local views associated with these numbers.
- $H(v)$ is the *history* of the vertex v. If at some computation step $(n, N, M, a) \in H(v)$ then it means that at some previous step the vertex v had a confidence level equal to a for the value M.

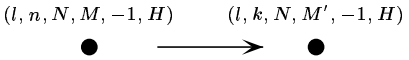
The first computation step $\mathcal{S}_0$ replaces just the initial label $\lambda(v)$ by $(\lambda(v), 0, \emptyset, \emptyset, -1, \emptyset)$. The following four rules mimic the rules of the enumeration algorithm:

$\mathcal{S}_1$:



If $M_2 \setminus M_1 \neq \emptyset$ then $M_1' := M_1 \cup M_2$.

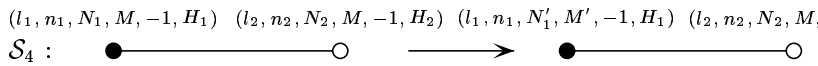
$\mathcal{S}_2$:



If $n = 0$ or there exists $(n, \ell', K') \in M$ such that $(\ell, \Pi_1(N)) \prec (\ell', K')$ then $k := 1 + \max\{n' \mid \exists (n', \ell', K') \in M\}$ and $M' := M \cup \{(k, \ell, \Pi_1(N))\}$.

$\mathcal{S}_3$:

If $n > 0$, $\Pi_1(N_1) = \Pi_1(N_2)$ and $\forall (n, \ell', K') \in M$, $(\ell', K') \preceq (\ell, \Pi_1(N_1))$ then $k := 1 + \max\{n' \mid \exists (n', \ell', K') \in M\}$, $N_1' := N_1 \setminus \{(n_1, a) \in N_1 \mid n_1 < n\} \cup \{(n, -1)\}$ and $M' := M \cup \{(k, \ell, \Pi_1(N_1'))\}$.

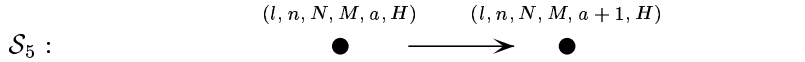


If $n_1 > 0$, $n_2 > 0$, $n_1 \neq n_2$, $\forall (n_1, \ell_1', K_1') \in M$, $(\ell_1', K_1') \preceq (\ell_1, \Pi_1(N_1))$, $\forall (n_2, \ell_2', K_2') \in M$, $(\ell_2', K_2') \preceq (\ell_2, \Pi_1(N_2))$, and $(n_2, -1) \notin N_1$ then $N_1' := N_1 \setminus \{(n', -1) \in N_1 \mid n' < n_2\} \cup \{(n_2, -1)\}$ and $M' := M \cup \{(n_1, \ell_1, N_1')\}$.

The fifth rule says that if a vertex v detects that all the neighbours it knows have a confidence level $a \geq a(v)$ then it can increment its own confidence level.

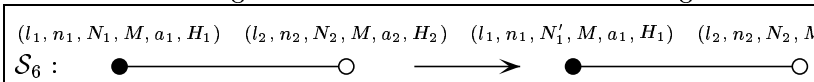
To define this rule we need some additional notations. Given a mailbox content M, for each $n > 0$ we define $\pi_n(M)$ as the set of all triples $(n, \ell, N) \in M$ with the first component n. For each non empty set $\pi_n(M)$ we conserve in the mailbox only the triple (n, ℓ, N) with the greatest couple (n, N) for the order $\prec$. This operation gives a new mailbox content that we shall note $u(M)$.

The next step consists in defining a graph G_M. If there exist $(n_1, \ell_1, N_1), (n_2, \ell_2, N_2) \in u(M)$ such that $(n_2, \ell_2) \in N_1$ and $(n_1, \ell_1) \notin N_2$ then we set $G_M = (\emptyset, \emptyset)$. Otherwise, G_M is the graph such that $V(G_M) = \{n \mid (n, \ell, N) \in u(M)\}$ and $E(G_M) = \{\{n_1, n_2\} \mid \exists (n_1, \ell_1, N_1), (n_2, \ell_2, N_2) \in u(M), (n_2, \ell_2) \in N_1$ and $(n_1, \ell_1) \in N_2\}$. The labelling of G_M is inherited from the set M: for $(n, \ell, N) \in u(M)$, $\lambda_M(n) = \ell$. We will denote by $\mathbf{G}_M = (G_M, \lambda_M)$ the corresponding labelled graph.



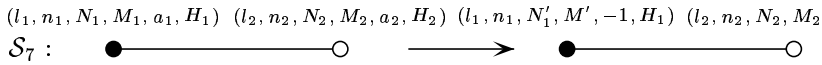
This rule applies whenever $\forall (n, \ell', N') \in M, (\ell', N') \preceq (\ell, \Pi_1(N))$, $\mathbf{G}_M \in \mathcal{S}_{\mathbf{G}}$, and $\forall (n', a') \in N, a \leq a'$, and $a \leq |V(\mathbf{G})| + 1$.

The sixth rule enables a node v to update its knowledge of the confidence level of one of its neighbour if the confidence level of this neighbour has increased.



If $a_1 \geq 0$, $\forall (n_2, \ell_2', N_2') \in M, (\ell_2', N_2') \preceq (\ell_2, \Pi_1(N_2))$, and there exists $(n_2, a) \in N_1$ such that $a_2 > a$ then $N_1' := N_1 \setminus \{(n_2, a)\} \cup \{(n_2, a_2)\}$.

The rule $\mathcal{S}_7$ enables a vertex v to change the value of its mailbox M whenever there exists a neighbour v' that used to have a confidence level a according to M such that $a \geq a(v) - 1$ and such that its mailbox has changed. If a vertex changes its mailbox, then it modifies also its history $H(v)$, so as to remember its former confidence level.



If $\exists (n, l, N) \in M_2 \setminus M_1$ and either $a_1 = 0$ or $(a_1 \geq 0$ and $\exists (n, N, M_1, a) \in H_2, \exists (n, a') \in N_1, a \geq a')$ then $N_1' := \{(n', -1) \mid \exists (n, a) \in N_1\}$, $M' := M_1 \cup M_2$ and $H_1' := H_1 \cup \{(n_1, N_1, M_1, a_1)\}$.

4.2 Correctness of the Election Algorithm

In the following $(\lambda(v), n_i(v), N_i(v), M_i(v), a_i(v), H_i(v))$ will stand for the label of the vertex v after the ith computation step of the election algorithm. The most important property of the algorithm is given in the following proposition. Roughly speaking it states that if the confidence level of vertex v is $|V(G)| + 2$ then $\mathbf{G}$ contains a submersion of $\mathbf{G}_{M(v)}$.

Proposition 10. *If there exists a vertex $v_0 \in V(\mathbf{G})$ and a step i_0 such that $a_{i_0}(v_0) = |V(\mathbf{G})| + 2$, $\mathbf{G}$ contains a submersion $\mathbf{H}$ of $\mathbf{G}_{M_{i_0}(v_0)}$ and for every step $i \geq i_0$ and for every vertex $v \in V(\mathbf{H})$, $M_i(v) = M_{i_0}(v_0)$.*

From Proposition 10 we deduce that if the conditions of Proposition 9 are satisfied then adding the following rule $\mathcal{S}_8$ allows to elect a unique vertex of $\mathbf{G}$: $\mathcal{S}_8$: the label (ℓ, n, N, M, a, H) such that $n = \max\{n \in C_{\mathbf{G}_M}(\mathbf{G})\}$ and $a = |V(\mathbf{G})|+2$ is replaced by *elected*. The last two rules serve to propagate the information that there is an elected vertex: $\mathcal{S}_9$ allows to transform a label of a vertex with an *elected* neighbour to *non-elected* and $\mathcal{S}_{10}$ propagates the *non-elected* label to all neighbours which are neither *elected* nor *non-elected*.

Summarising we get:

Theorem 11. *There exists an election algorithm over a given graph $\mathbf{G}$ using cellular edge local computations if and only if the following conditions are satisfied:*

1. *for every $\mathbf{H} \in \mathcal{S}_{\mathbf{G}}$, $C_{\mathbf{H}}(\mathbf{G}) \neq \emptyset$,*
2. *there do not exist two disjoint subgraphs $\mathbf{G}_1$ and $\mathbf{G}_2$ of $\mathbf{G}$ such that $\mathbf{G}_1$ (resp. $\mathbf{G}_2$) is a submersion of a graph $\mathbf{H}_1 \in \mathcal{S}_{\mathbf{G}}$ (resp. $\mathbf{H}_2 \in \mathcal{S}_{\mathbf{G}}$).*

5 Examples

If we assume that nodes of a graph $\mathbf{G}$ have unique identifiers then $\mathbf{G}$ is a submersion-minimal graph and the knowledge of its size allows an election.

5.1 Trees, Grids and Complete graphs

Consider an unlabelled tree T. Since we can colour each tree T with just two colours, if T has at least 2 vertices such colouring yields a submersion of T into the graph K_2 with two vertices and one edge between them. Such a submersion is non trivial if T has at least 3 vertices. Therefore for such trees there does not exist a naming algorithm using cellular edge local computations.

If $\varphi_1 : T \to K_2$ is a submersion (colouring) of T then exchanging the two colours we get another submersion φ_2 and if T has at least three vertices then for each colour $k \in V(K_2)$ at least one of the sets $\varphi_i(k)$, $i = 1, 2$, has cardinality ≥ 2. Consequently, the election problem cannot be solved for trees with more than 2 vertices.

For the same reasons, square grids, which are also connected and colourable with two colours, do not admit either naming or election algorithms in our model.

Complete graphs are submersion-minimal and therefore admit both naming and election in our model.

5.2 Rings with a Prime Size

First note the following fact:

Proposition 12. *An unlabelled ring of size p is submersion-minimal if and only if p is prime.*

Therefore prime size rings allow both naming and election. This is a quite interesting corollary of our general conditions since our model is the weakest among graph relabelling systems, with the bare minimal synchronisation power. Moreover, contrary to some other algorithms on rings, our enumeration algorithm does not need any sense of direction for computing agents.

References

1. D. Angluin. Local and global properties in networks of processors. In *Proceedings of the 12th Symposium on Theory of Computing*, pages 82–93, 1980.
2. H. Attiya and J. Welch. *Distributed computing: fundamentals, simulations, and advanced topics.* McGraw-Hill, 1998.
3. Paolo Boldi, Bruno Codenotti, Peter Gemmell, Shella Shammah, Janos Simon, and Sebastiano Vigna. Symmetry breaking in anonymous networks: Characterizations. In *Proc. 4th Israeli Symposium on Theory of Computing and Systems*, pages 16–26. IEEE Press, 1996.
4. J. Chalopin and Y. Métivier. Election and local computations on edges (*extended abstract*). In *Proc. of Foundations of Software Science and Computation Structures, FOSSACS'04*, number 2987 in LNCS, pages 90–104, 2004.
5. E. Godard, Y. Métivier, and A. Muscholl. Characterization of Classes of Graphs Recognizable by Local Computations. *Theory of Computing Systems*, (37):249–293, 2004.
6. G. LeLann. Distributed systems: Towards a formal approach. In B. Gilchrist, editor, *Information processing'77*, pages 155–160. North-Holland, 1977.
7. N. A. Lynch. *Distributed algorithms.* Morgan Kaufman, 1996.
8. A. Mazurkiewicz. Distributed enumeration. *Inf. Processing Letters*, 61:233–239, 1997.
9. B. Szymanski, Y. Shy, and N. Prywes. Terminating iterative solutions of simultaneous equations in distributed message passing systems. In *Proc. of the 4th Symposium of Distributed Computing*, pages 287–292, 1985.
10. G. Tel. *Introduction to distributed algorithms.* Cambridge University Press, 2000.
11. M. Yamashita and T. Kameda. Computing on anonymous networks: Part i - characterizing the solvable cases. *IEEE Transactions on parallel and distributed systems*, 7(1):69–89, 1996.