Fundamenta Informaticae 74 (2006) 85–114
IOS Press

Local Computations in Graphs: The Case of Cellular Edge Local Computations

Jérémie Chalopin[*], **Yves Métivier**
LaBRI
ENSEIRB, Université Bordeaux 1
351 cours de la Libération, 33405 Talence, France
{chalopin,metivier}@labri.fr

Wiesław Zielonka
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 characterize the graphs for which two important distributed algorithmic problems are solvable in this model : naming and election.

Keywords: Distributed Computing, Local Computations, Election, Naming, Submersion

1. Introduction

Various models of local computations on graphs provide an elementary and convenient framework to study basic algorithmic problems of distributed computing. In these models we have at our disposal only bare synchronisation primitives and the corresponding local computation steps. It turns out that for many basic algorithmic problems arising in distributed computing such simple models are sufficient, they allow

[*]Address for correspondence: LaBRI, ENSEIRB, Université Bordeaux 1, 351 cours de la Libération, 33405 Talence, France

either to formulate an algorithm solving the problem or to show formally that the problem is not solvable in the given context.

The relative simplicity of the local graph computation models facilitates the conception of algorithms and suitable combinatorial structures which subsequently can be translated and applied in a more realistic but also more involved setting. Such models give us an insight which is difficult to obtain in more elaborate models.

Local graph computations often allow to delimit precisely the borderline between positive and negative results in distributed computing. It is clear that the possibility of solving a particular problem for a given class of networks depends on the power of the synchronisation primitives and on the initial knowledge available to the computational agents. Better comprehension of all these factors enhances our understanding of basic distributed algorithmic problems.

Another point of view is to consider local graph computation as a kind of a higher-level language for designing algorithms in asynchronous distributed systems and proving their correctness.

As it is well established in the domain of distributed computing, some algorithmic problems like: election, naming, termination detection, network topology recognition constitute basic building blocks for many other algorithms. Yamashita and Kameda [19, 20], Boldi et al. [3], Mazurkiewicz [15, 16], Godard et al. [11, 12], Chalopin et al. [4, 5, 6] characterize graphs in which election or naming are possible for different models of distributed computations.

The local graph computation model that we examine in our paper is the most elementary one and all the other local computation models considered in the papers cited above [4, 5, 15, 16] can simulate it trivially. In fact this is the weakest possible model allowing any synchronisation at all. In this model 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 focus our attention on two important algorithmic problems: the naming and the election problems. Although these two problems are often equivalent in other models this is not the case in our model.

To characterize the graphs where we can solve both problems, we find suitable graph homomorphisms that enable us to formulate conveniently the necessary conditions. This step is similar to Angluin [1], but in our case the relevant homomorphisms are graph submersions. The presented conditions turn out to be also sufficient: algorithms, inspired by Mazurkiewicz [15], are given, that enable us 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 v. An elementary computation step will be represented by relabelling rules of the form given schematically in Figure 1. 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 the other 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 in figures), the neighbour vertex used to match the rule is called *passive* (and marked as unfilled in figures). All the other vertices of G not participating in such elementary relabelling step are called *idle*. 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

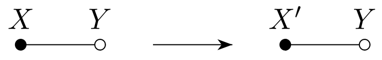


Figure 1. Graphical form of a rule for cellular edge local computations.

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. The formal definitions of the model follow in Section 2 and Section 3.

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 [13]. 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 centralized decisions, to initialize some other activity, to centralize 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 aim of a naming algorithm is to arrive at a final configuration where all processors have unique identities. Again this is an essential prerequisite to many other distributed algorithms which 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 $\{1, 2, \ldots, |V(G)|\}$.

In our setting, a distributed algorithm terminates if the network is in such a state that no relabelling rule can be applied, but it does not mean that the processes are aware that the computation has terminated. We say that we can solve a problem with termination detection on a graph G if there exists an algorithm $\mathcal{A}$ that solves the problem on G such that in the final state, at least one vertex is aware that no relabelling rule can be applied in the graph.

The naming and the election problems are often equivalent for various computational models [4, 5, 6, 15, 16, 19, 20], 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 characterization; unfortunately a similar characterization 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 characterization of labelled graphs for which naming and election are possible. Both problems are solved constructively, we present naming and election algorithms that work correctly for all labelled graphs where these problems are solvable. Our naming algorithm is a nice adaptation of the algorithm of Mazurkiewicz [15], whereas our election algorithm uses also an adaptation of the algorithm of Szymanski et al. [17] which is usually used to detect the termination of an algorithm in the message passing model.

1.4. Related Works

The election problem was already studied in a great variety of models [2, 14, 18]. The proposed algorithms depend on the type of the basic computation steps, they work correctly only for a particular type of network topology (tree, grid, torus, ring with a known prime number of vertices etc.) or under the assumption that some initial extra knowledge is available to processors.

Various local computation models studied in the literature are characterized by the relabelling rules that they use. Figure 2 presents schematically such rules and their hierarchy in terms of the computational power. Characterizations of graphs where naming and election can be solved exist for each of these models, except for the model (1) that is studied in our paper.

Yamashita and Kameda [19] consider the model where, in each step, one of the vertices, depending on its current label, either changes its state, 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 vertices having the same view. The view from a vertex v of a graph G is an infinite labelled tree rooted in v obtained by considering all labelled walks in G starting from v. This message passing model is strictly less powerful than the model (6) in Figure 2 but its computational power is not comparable with the computational power of the models $(1), (2), (3), (4), (5)$ in Figure 2. In [20], Yamashita and Kameda study the importance of the port labelling for the election problem in this message passing model. From the results of Boldi et al. [3], one can obtain different characterizations for the different models considered in [20], based on fibrations and coverings of directed graphs.

In [6], Chalopin and Métivier consider also the message passing model where there exists a port numbering, and they give an encoding of the basic events by means of local computations on arcs. This encoding allows to give a new characterization of networks for which there exists an election algorithm in the message passing model. This characterization is based on symmetric coverings of directed graphs.

Mazurkiewicz [15] considers the asynchronous computation model where in one computation step labels are modified on a subgraph consisting of a node and its neighbours, according to rules depending on this subgraph only. This is the model (7) of Figure 2. Mazurkiewicz's characterization of the graphs where enumeration/election are possible is based on the notion of unambiguous graphs and may be formulated equivalently using coverings [10]. He gives a nice and simple enumeration algorithm for the graphs that are minimal for the covering relation, i.e., which can cover only themselves.

Boldi et al. [3] consider a model where the network is a directed multigraph G. They consider models where the arcs can be labelled or not. 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 generalizations of coverings. Boldi et 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 characterization, they use the same mechanism as Yamashita and Kameda: each node computes its own view and next the node

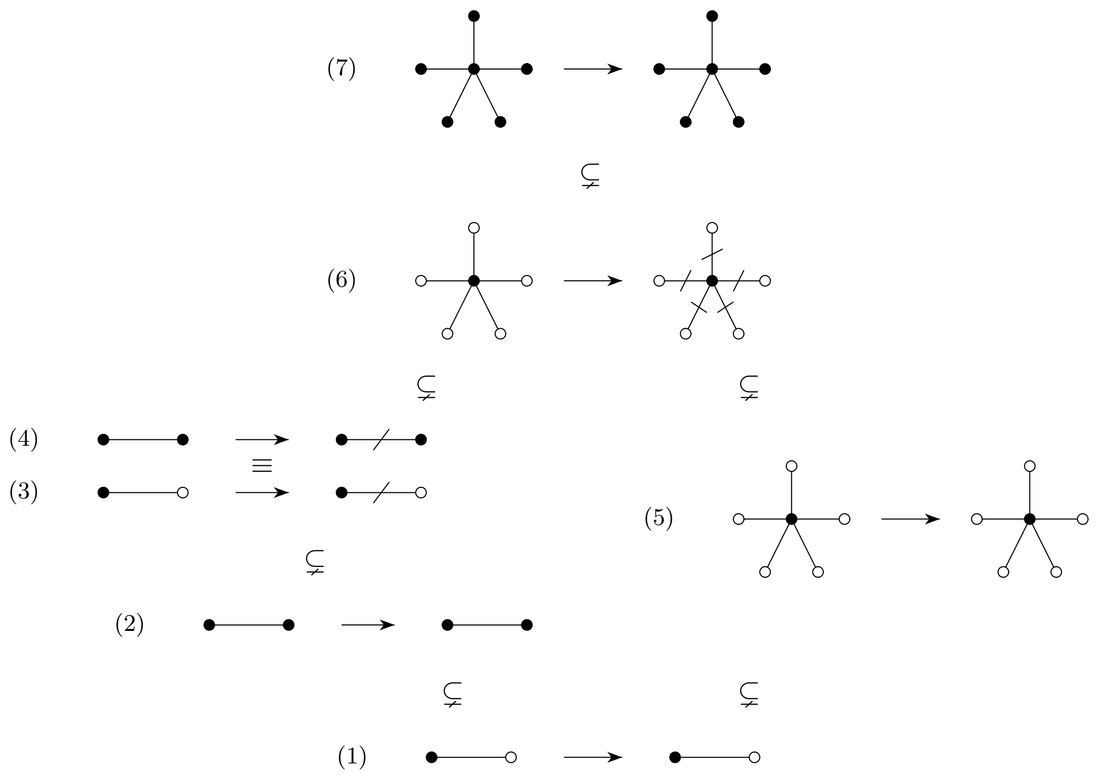


Figure 2. A hierarchy between the different models studied by Boldi et al., Chalopin et al. and Mazurkiewicz. The black vertices are active : their labels can change when the rule is applied. The white vertices are passive : their labels enable to apply the relabelling rule but they do not change. In the models where the edges are marked, the edges are labelled and the relabelling rules can modify their labels, whereas it is not the case for the other models. The inclusion $r_1 \subsetneq r_2$ between the rules r_1 and r_2 means that r_2 can simulate r_1 but not vice versa, i.e., r_2 has a greater computational power than r_1. The model (3) and (4) have the same computational power, and this is denoted by the symbol $\equiv$. The computational power of the model (5) is incomparable with the power of the models $(2), (3)$ and (4).

with the weakest view is elected. In the different models they consider, naming and election are not always equivalent. From the work of Boldi et al., we can easily deduce characterizations of the graphs where election is possible for the models (5) and (6) of Figure 2.

In [5], the models (3), (4) and (6) of Figure 2 were examined. Note that, contrary to the model we examine in the present paper, all these models allow edge labelling. In [5], it is shown that the models (3) and (4) are equivalent in terms of computational power but (6) has a strictly greater power. It turns out that for all these three models, election and naming over a given graph G are equivalent. In [5], it is proved that for the models (3), (4) and (6) 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 should note that the model (4) was also examined by Mazurkiewicz [16] who gives an equivalent characterization based on equivalence relations over graph vertices and edges. Let us note by the way that although the model studied in the present paper and the model (3) seem to be very close, the graphs for which the naming problem and the election problem can be solved are very different for both models. The intuitive reason is that if we allow to label the edges as in (3) then each processor can subsequently consistently identify the neighbours. On the other hand, in the model (1) that we examine here, where edges are not labelled, a vertex can never know if it synchronizes with the same neighbour as previously or with another one.

In [4], the model (2) of Figure 2 is studied: in one computation step, two neighbours modify simultaneously their labels according only to their states (the edges are not labelled). In this model, the graphs admitting both naming and election algorithms are pseudo-covering-minimal graphs. (A graph G is a pseudo-covering of H if there exists a homomorphism φ from G onto H and a partial graph G' of G such that G' is a covering of H via the restriction of φ to G').

The study of local computations uses various locally constrained graph homomorphisms. Some properties and a complexity classification may be found in [8, 9].

The paper is organised as follows. Section 2 reviews the basic definitions of labelled graphs and submersions. In Section 3 we give the definition of cellular edge local computations as well as the relation between such computations and submersions. In Section 4 we prove that there is no naming (enumeration) algorithm for graphs which are not submersion-minimal. Then we give a naming (an enumeration) algorithm for submersion-minimal graphs. In Section 5 we give a graph for which the election problem can be solved but not the enumeration problem. Then, as in Section 4, we characterize graphs which admit an election algorithm. Section 6 presents some examples of graphs that admit naming or election algorithms. Section 7 (conclusion) presents some open problems.

A preliminary version of this paper has been published in [7].

2. Preliminaries

2.1. Graphs and Labelled Graphs

We consider finite, undirected, connected graphs $G = (V(G), E(G))$ with vertices $V(G)$ and edges $E(G) \subset V(G) \times V(G)$ 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 incident to v. The degree of a vertex v, denoted $d_G(v)$, is the number of edges incident with v.

A path of length p is a sequence $(v_0, v_1, v_2, \ldots, v_p)$ of vertices such that for every $0 \leq i < p$, $\{v_i, v_{i+1}\} \in E(G)$. This path is simple if all vertices are pairwise distinct.

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 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.

Throughout the paper we will consider graphs with vertices labelled with labels from a recursive label set L. A graph labelled over L is an ordered pair $\mathbf{G} = (G, \lambda)$, where G is the underlying unlabelled graph and $\lambda\colon V(G) \to L$ is the (vertex) labelling function.

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.

For any finite 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 $\{1, 2, \ldots, q\}$.

2.2. Submersions

Graph submersions, i.e., locally surjective graph homomorphisms, are the basic tool allowing to formulate necessary conditions for naming and election problems in our paper:

Definition 2.1. A graph G is a submersion of a graph H via a homomorphism $\gamma\colon G \to H$ if for each $v \in V(G)$, 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. Finally, 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.

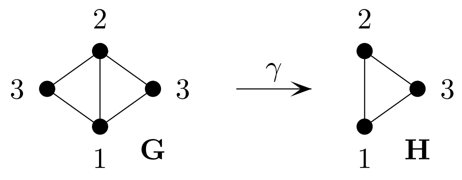


Figure 3. The labelled graph $\mathbf{G}$ is a submersion of $\mathbf{H}$ via the mapping γ which maps each vertex of $\mathbf{G}$ labelled i to the unique vertex of $\mathbf{H}$ with the same label i. This submersion is proper and the graph $\mathbf{H}$ is itself submersion-minimal.

The following elementary proposition characterizes submersion-minimal graphs as the graphs $\mathbf{G}$ for which there does not exist any colouring function using less than $|V(G)|$ colours and satisfying some additional properties: two vertices having distinct initial labels have distinct colors, two adjacent vertices have distinct colors, two vertices having the same color "see" the same colors in their neighbourhood.

Proposition 2.1. A graph $\mathbf{G} = (G, \lambda)$ is submersion-minimal if and only if there is no colouring function f from $V(G)$ to a set of colours C with $|C| < |V(G)|$ satisfying the following conditions:

- $\forall v, v' \in V(G), f(v) = f(v') \implies \lambda(v) = \lambda(v')$;
- $\forall v \in V(G), \forall v' \in N_G(v), f(v) \neq f(v')$;
- $\forall v, v' \in V(G)$ such that $f(v) = f(v'), \forall w \in N_G(v), \exists w' \in N_G(v') : f(w) = f(w')$.

Example 2.1. A complete graph $\mathbf{G}$ cannot be coloured with less than $|V(G)|$ colours and therefore is submersion-minimal.

3. Cellular Edge Local Computations

In this section, after giving the formal definition of cellular edge local computations, we study their relations with submersions.

3.1. Definitions

Let us recall that informally cellular edge local computations are computations that are realized by means of the rules of the form presented in Figure 1: at each step the label of one vertex is modified according to a rule depending on the label of this vertex and the label of one of its neighbours. The more formal framework is the following.

Let $\mathcal{G}_L$ be the class of L-labelled graphs. Then any binary relation $\mathcal{R} \subseteq \mathcal{G}_L \times \mathcal{G}_L$ on $\mathcal{G}_L$ is called a *graph rewriting relation*. We assume that $\mathcal{R}$ is closed under isomorphism, i.e., if $\mathbf{G}\ \mathcal{R}\ \mathbf{G}'$ and $\mathbf{H} \simeq \mathbf{G}$ then $\mathbf{H}\ \mathcal{R}\ \mathbf{H}'$ for some labelled graph $\mathbf{H}' \simeq \mathbf{G}'$. In the remainder of the paper $\mathcal{R}^*$ stands for the reflexive and transitive closure of $\mathcal{R}$. The labelled graph $\mathbf{G}$ is $\mathcal{R}$*-irreducible* (or just irreducible if $\mathcal{R}$ is fixed) if there is no $\mathbf{G}'$ such that $\mathbf{G}\ \mathcal{R}\ \mathbf{G}'$.

Definition 3.1. Let $\mathcal{R} \subseteq \mathcal{G}_L \times \mathcal{G}_L$ be a graph rewriting relation.

1. $\mathcal{R}$ is a *relabelling relation* if whenever two labelled graphs are in relation then the underlying unlabelled graphs are equal, i.e.,:

$$\mathbf{G}\ \mathcal{R}\ \mathbf{H} \text{ implies that } G = H.$$

2. $\mathcal{R}$ is *cellular* if it can only modify the label of only one vertex, i.e., $(G, \lambda)\ \mathcal{R}\ (G, \lambda')$ implies that there exists a vertex $v \in V(G)$ such that

$$\lambda(x) = \lambda'(x) \text{ for every } x \neq v.$$

The next definition states that a cellular local relabelling relation $\mathcal{R}$ is *edge locally generated* if the applicability of any relabelling depends only on the labels of the vertices incident to an edge.

Definition 3.2. Let $\mathcal{R}$ be a relabelling relation. Then $\mathcal{R}$ is *cellular edge locally generated* if it is cellular and the following is satisfied: for all labelled graphs (G, λ), (G, λ'), (H, η), (H, η') and all edges $\{v_1, v_2\} \in E(G)$ and $\{w_1, w_2\} \in E(H)$, the following three conditions:

1. $\lambda(v_1) = \eta(w_1)$, $\lambda(v_2) = \eta(w_2)$ and $\lambda'(v_1) = \eta'(w_1)$,
2. $\lambda(v) = \lambda'(v)$, for all $v \neq v_1$,
3. $\eta(w) = \eta'(w)$, for all $w \neq w_1$,

imply that $(G, \lambda)\ \mathcal{R}\ (G, \lambda')$ if and only if $(H, \eta)\ \mathcal{R}\ (H, \eta')$.

We only consider recursive relabelling relations. The purpose of all assumptions about recursiveness done throughout the paper is to have "reasonable" objects w.r.t. the computational power. By definition, *cellular edge local computations on graphs* are computations on graphs corresponding to cellular edge locally generated relabelling relations.

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.

A cellular edge locally generated relabelling relation can be described by a recursive set of relabelling rules of the type represented in Figure 1. And equivalently, such a set of rules induces a cellular edge locally generated relabelling relation. Thus, slightly abusing the notation, $\mathcal{R}$ will stand both for a set of rules and the induced relabelling relation over labelled graphs.

3.2. Submersions and Cellular Edge Local Computations

We now present the fundamental lemma connecting submersions and cellular edge local computations. This is a counterpart of the lifting lemma of Angluin [1] adapted to submersions.

Lemma 3.1. (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 relabelling step. Let $(G, \lambda), (H, \nu)$ be two graphs such that (G, λ) is a submersion of (H, ν) via φ. Suppose that the relabelling step is applied to the active vertex $w \in V(H)$ and to the passive vertex $w' \in V(H)$ yielding a new labelling ν' on H.

For every vertex $v \in \varphi^{-1}(w)$, since the homomorphism φ is locally surjective, there exists $v' \in \varphi^{-1}(w') \cap N_G(v)$. Since the vertices of $\varphi^{-1}(w)$ are pairwise non-adjacent, $v' \notin \varphi^{-1}(w)$ and therefore, we can apply the rule to every vertex $v \in \varphi^{-1}(w)$. 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. □

This is depicted in the following diagram:

$$\begin{tikzcd}
\mathbf{G} \arrow[r, "\mathcal{R}^*"] \arrow[d, "\text{submersion}"'] & \mathbf{G}' \arrow[d, "\text{submersion}"] \\
\mathbf{H} \arrow[r, "\mathcal{R}^*"'] & \mathbf{H}'
\end{tikzcd}$$

4. Enumeration and Naming Problems

This section presents a characterization of graphs which admit an enumeration or a naming algorithm. First we prove that there exist no naming and no enumeration algorithms on a graph $\mathbf{G}$ that use cellular edge local computations if the graph is not submersion-minimal. The proof is analogous to that of Angluin [1]. Then we give and we prove an enumeration (a naming) algorithm for submersion-minimal graphs.

4.1. Impossibility results for enumeration and naming

Proposition 4.1. Let $\mathbf{G}$ be a labelled graph which is not submersion-minimal. Then there is no naming and no enumeration algorithm for $\mathbf{G}$ using cellular edge local computations.

Proof:
Let $\mathbf{H}$ be a labelled graph not isomorphic to $\mathbf{G}$ and 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 3.1, 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 vertices $v, v' \in V(G)$ such that $\lambda'(v) = \lambda'(v')$. Consequently, $\mathcal{R}$ solves neither the naming nor the enumeration problem on $\mathbf{G}$. □

4.2. An Enumeration Algorithm

In this subsection, 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 unique identity which is a number between 1 and $|V(G)|$. The vertex chooses a number and gathers the information about the numbers chosen by its neighbours. Then, it broadcasts its number with its *local view* (which is the set of the numbers of its neighbours). If a vertex u discovers the existence of another vertex v with the same number then it should decide if it changes its identity. To this end it compares its local view with the local view of v. If the label of u or the local view of u is "weaker", then u picks another number — its new temporary identity — and broadcasts it again with its local view. At the end of the computation, if the graph is submersion-minimal then every vertex will have a unique number .

4.2.1. Labels

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 always be 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, i.e., the numbers and the local views of the vertices of the network.

In many distributed algorithms, it is required that processors have unique identities, and that each processor knows its own identity initially. Thus in our framework the labelling λ of the graph may encode the identities of the vertices (thus $\forall v, v' \in V(G), \lambda(v) \neq \lambda(v')$). On the other hand, if we deal with anonymous systems, i.e., all the processes have initially the same name, the labelling λ will be such that $\forall v, v' \in V(G), \lambda(v) = \lambda(v')$. More generally, the labelling may encode any initial processor knowledge. Examples of such knowledge include: no initial knowledge, the number of processors, the diameter of the graph and the topology. Sometimes k processors have a given label: there are k distinguished vertices. Thus this modelization encompasses: topological restrictions (tree, complete networks, ...), topological knowledges (size, diameter, ...), and local knowledges (identities, degrees, ...).

4.2.2. An Order on Local Views

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 [15]. The algorithm described below is such that the local view of any vertex cannot decrease during the computation.

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 $N_1 \prec N_2$ if and only if one of the following conditions hold:

- $k < l$ and for all i, $1 \leq i \leq k$, $n_i = m_i$,
- $n_i < m_i$ 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 the paper that the set of initial labels L is totally ordered by $<_L$. 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 $\prec$.

4.2.3. The Relabelling Rules

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 [15].

The first rule $\mathcal{M}_1$ enables a vertex to update its mailbox by looking at the mailbox of one of its neighbours:

$\mathcal{M}_1$:

$$\begin{tikzcd} \overset{(\ell_1, n_1, N_1, M_1)}{\bullet} \arrow[r, dash] & \overset{(\ell_2, n_2, N_2, M_2)}{\circ} \arrow[r] & \overset{(\ell_1, n_1, N_1, M_1')}{\bullet} \arrow[r, dash] & \overset{(\ell_2, n_2, N_2, M_2)}{\circ} \end{tikzcd}$$

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 synchronization 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.

$\mathcal{M}_2$:

$$\begin{tikzcd} \overset{(\ell, n, N, M)}{\bullet} \arrow[r] & \overset{(\ell, k, N, M')}{\bullet} \end{tikzcd}$$

If $(n = 0)$ or $(\exists (n, \ell', N') \in M$ and $(\ell, N) \prec (\ell', N'))$

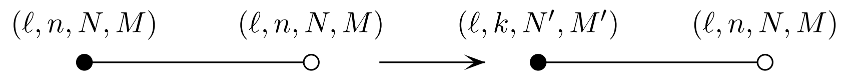
then $k := 1 + \max\{n' \mid \exists (n', \ell', N') \in M\}$ and

$M' := M \cup \{(k, \ell, N)\}$.

(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). This relabelling step can be applied only if the rule $\mathcal{M}_2$ cannot be applied by v or v'. 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.

$\mathcal{M}_3$:



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 synchronizes with a neighbour v' and observes that the current identity number $n(v')$ of v' does not belong to its 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 synchronizes 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 synchronized 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, it should delete the old identity number of v' from its local view. 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 synchronization 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)synchronizing with all such neighbours. The conditions of the if-part imply that a vertex has to update its local view and that neither $\mathcal{M}_1$ nor $\mathcal{M}_2$ and nor $\mathcal{M}_3$ are applicable. This rule transmits the name of a vertex to the view of an adjacent vertex and updates its mailbox.

$\mathcal{M}_4$:

$$\begin{tikzcd} \overset{(\ell_1,n_1,N_1,M)}{\bullet} \arrow[r, no head] & \overset{(\ell_2,n_2,N_2,M)}{\circ} \arrow[r] & \overset{(\ell_1,n_1,N'_1,M')}{\bullet} \arrow[r, no head] & \overset{(\ell_2,n_2,N_2,M)}{\circ} \end{tikzcd}$$

If $(n_1 > 0, n_2 > 0, n_1 \neq n_2)$ and
$\forall (n_1, \ell'_1, N'_1) \in M, (\ell'_1, N'_1) \preceq (\ell_1, N_1)$ and
$\forall (n_2, \ell'_2, N'_2) \in M, (\ell'_2, N'_2) \preceq (\ell_2, N_2)$ and
$(n_2 \notin N_1)$

then $(N'_1 := N_1 \setminus \{n' \in N_1 \mid n' < n_2\}) \cup \{n_2\}$ and
$M' := M \cup \{(n_1, \ell_1, N'_1)\}$.

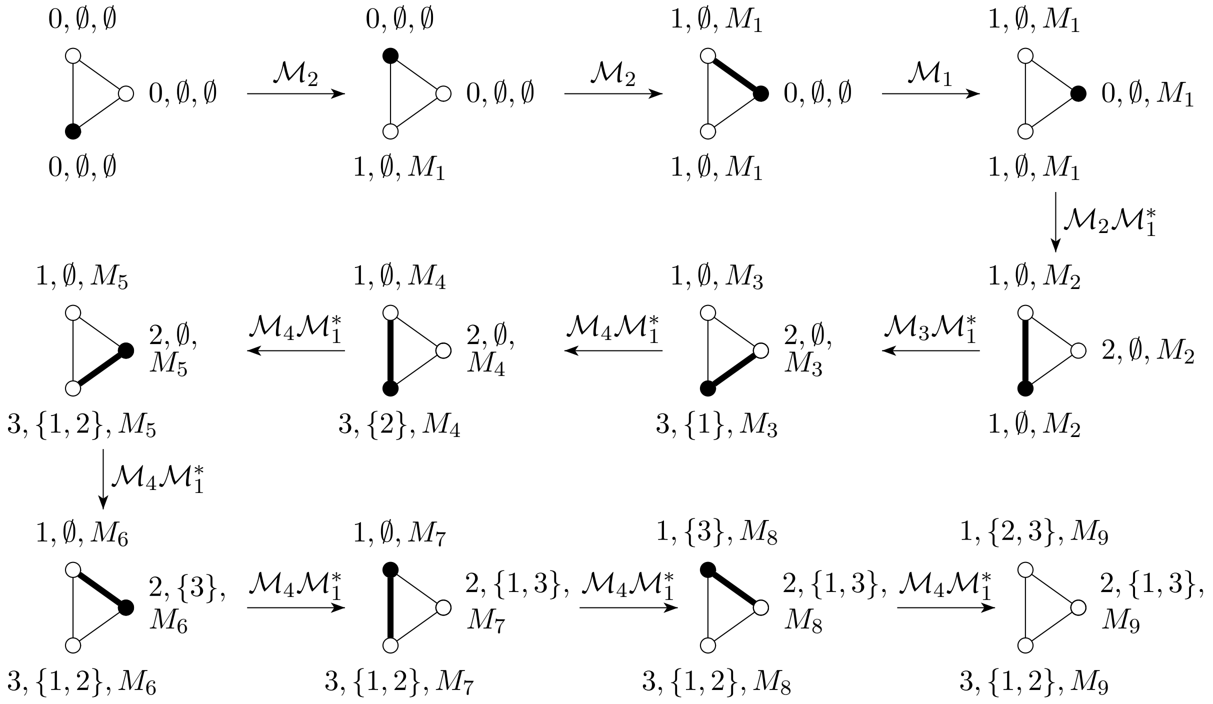


Figure 4. An execution of the algorithm $\mathcal{M}$ on K_3 where all the vertices have the same initial label (that is not mentioned, for sake of clarity). At each step, the vertex filled with black is active and apply the rule $\mathcal{M}_i$ along the thick edge that labels the following relabelling step. If a relabelling step is called $\mathcal{M}_i\mathcal{M}_1^*$, it means that we first apply the rule $\mathcal{M}_i$ on the active vertex and then, we apply the rule $\mathcal{M}_1$ as many times as possible. For sake of clarity, we do not have explicit the content of the mailbox M_i at each step: it contains all the (n, N) that appears on any vertex of the graph earlier in the execution.

An example of an execution of the algorithm $\mathcal{M}$ on the complete graph K_3 with three vertices is presented in Figure 4.

4.3. Correctness of the Enumeration Algorithm

Let **G** be a simple labelled graph. In the following, i is an integer denoting a computation step. Let $(\lambda(v), (n_i(v), N_i(v), M_i(v))$ be the label of the vertex v after the ith step of the computation of the algorithm $\mathcal{M}$ given above. We present here some properties satisfied by each execution of the algorithm.

The following lemma, which can be proved easily by induction on the number of steps, recapitulates basic labelling properties.

Lemma 4.1. For each vertex v and each step i,

1. $n_i(v) \neq 0 \implies (n_i(v), \lambda(v), N_i(v)) \in M_i(v)$,
2. $\forall n' \in N_i(v), n' \neq 0$ and $\exists \ell' \in L, \exists N' \in \mathcal{P}_{\text{fin}}(\mathbb{N}), (n', \ell', N') \in M_i(v)$,
3. $n_i(v) \notin N_i(v)$.

The algorithm has some remarkable monotonicity properties that are described in the following lemma.

Lemma 4.2. For each step i and each vertex v:

- $n_i(v) \leq n_{i+1}(v)$,
- $N_i(v) \preceq N_{i+1}(v)$,
- $M_i(v) \subseteq M_{i+1}(v)$.

Moreover, there exists at least one vertex v such that at least one of these inequalities/inclusions is strict for v.

Proof:
The property is obviously true for the vertices that are not involved in the rule applied at step i. It is easy to see that, for each vertex v, we always have $M_i(v) \subseteq M_{i+1}(v)$.

For each vertex v and each step i such that $n_i(v) \neq n_{i+1}(v)$, $n_{i+1}(v) = 1 + \max\{n_1; (n_1, \ell_1, N_1) \in M_i(v)\}$ and either $n_i(v) = 0 < n_{i+1}(v)$ or $(n_i(v), \lambda(v), N_i(v)) \in M_i(v)$ as shown in Lemma 4.1 and therefore $n_i(v) < n_{i+1}(v)$.

For each vertex v such that $N_i(v) \neq N_{i+1}(v)$, the rule $\mathcal{M}_3$ or $\mathcal{M}_4$ has been applied between v and one of its neighbours v'. For every $n \in N_i(v)$, if $n > n_{i+1}(v')$, $n \in N_{i+1}(v)$ and $n_{i+1}(v) \in N_{i+1}(v) \setminus N_i(v)$. Consequently, $N_i(v) \prec N_{i+1}(v)$.

For each step i, the rule applied at this step modifies the label of one vertex v and therefore one of these inequalities is strict for v. □

The local knowledge of a vertex v reflects to some extent some real properties of the current configuration. The two following lemmas enable us to prove that if a vertex v knows a number m (i.e., there exist ℓ, N such that $(m, \ell, N) \in M_i(v)$), then for each $m' \leq m$, there exists a vertex v' in the graph such that $n_i(v') = m'$. We first show that if v knows m there exists v' such that $n_i(v') = m$.

Lemma 4.3. For every $v \in V(G)$ and $(m, \ell, N) \in M_i(v)$, there exists a vertex $w \in V(G)$ such that $n_i(w) = m$.

Proof:
Assume that the number m is known by v and let $U = \{(u, j) \in V(G) \times \mathbb{N} \mid j \leq i, n_j(u) = m\}$. Consider the set $U' = \{(u, j) \in U \mid \forall (u', j') \in U, N_{j'}(u') \prec N_j(u)$ or $N_{j'}(u') = N_j(u)$ and $j' \leq j\}$. It is easy to see that there exists i_0 such that for each $(u, j) \in U', j = i_0$. Since $(m, \ell, N) \in M_i(v)$, neither U nor U' are empty.

If $i_0 < i$, there is at most one element (u, i_0) in U' because at each step the number of at most one vertex can change. The number $n(u)$ of the vertex u must have therefore changed at step $i_0 + 1$ but since at step $i_0 + 1$ vertex u has no neighbour with the same number, the rule $\mathcal{M}_3$ cannot be applied and by maximality, the rule $\mathcal{M}_2$ cannot be applied either. Consequently, there exists a vertex $w \in V(G)$ such that $n_i(w) = m$. □

In the following lemma, we show that if a vertex v knows an identity number m, then it knows all the numbers smaller than m.

Lemma 4.4. For every vertex $v \in V(G)$ and every step i such that $n_i(v) \neq 0$, given $(m', \ell', N') \in M_i(v)$, for every $1 \leq m \leq m'$, there exists $(m, \ell, N) \in M_i(v)$.

Proof:
We show this claim by induction on i. At the initial step the assertion is true. Suppose that it holds for $i \geq 0$.

If the rule $\mathcal{M}_4$ is applied and it modifies the label of a vertex v, for every $(m, \ell, N) \in M_{i+1}(v)$, there exists $(m, \ell, N') \in M_i(v)$ and the property holds. If the rule $\mathcal{M}_1$ is applied to a vertex v, depending on the label of one of its neighbours v', then $M_{i+1}(v) = M_i(v) \cup M_i(v')$ and by induction, the property holds. If the rule $\mathcal{M}_2$ is applied to a vertex v, $M_{i+1}(v) = M_i(v) \cup \{(n_{i+1}(v) = 1 + \max\{m \mid (m, \ell, N) \in M_i(v)\}, \lambda(v), N_i(v))\}$, and consequently for each $m \in M_{i+1}(v)$, the property is still true. For the same reasons, the property holds when the rule $\mathcal{M}_3$ is applied. □

From Lemmas 4.3 and 4.4, we can deduce that for each step, the identity numbers of all the vertices form either a set $[1, k]$ or a set $[0, k]$ with $k \leq V(G)$.

For each step i and each vertex v, if there exists $n' \in N_i(v)$, from Lemma 4.1, there exists v' such that $n_i(v') = n'$ and therefore $N(v)$ can only have a finite number of values and the same holds for $M(v)$. During the algorithm, the consecutive labellings of each vertex v form an increasing sequence, $(n_i(v), N_i(v), M_i(v))$, $i = 1, 2, \ldots$ and, for each i, at least one label strictly increases. Since the number of possible accessible labels is finite (but dependent on the size of the graph) the relation $\mathcal{M}$ is noetherian: the algorithm always terminates.

In the following lemma, it is shown that if a vertex v has an identity number n in its local view then either v has a neighbour v' such that $n(v') = n$ or the rule $\mathcal{M}_4$ can be applied to v and a neighbour v'.

Lemma 4.5. For every step i, for every vertex $v \in V(G)$ and every $n_0 \in N_i(v)$, there exists $v' \in N_G(v)$ such that either $n_i(v') = n_0$ or $n_i(v') > \max\{n \in N_i(v)\}$.

Proof:
Let i_0 be the last step in which n_0 has been added to $N(v) : \forall j \geq i_0, n_0 \in N_j(v)$ and $n_0 \notin N_{i_0 - 1}(v)$. There exists a vertex v' such that at the step i_0, the rule $\mathcal{M}_3$ or $\mathcal{M}_4$ is applied to v and v' and consequently,

$n_{i_0}(v') = n_0$ and from Lemma 4.1, there exists $(n_0, \ell, N) \in M_{i_0}(v')$. If $n_i(v') = n_{i_0}(v')$, the property is satisfied.

Otherwise, for every step j, we define $m_j(v) = \max\{n \in N_j(v)\}$. Clearly, $m_{i_0}(v) \geq n_0$ and there exists $(m_{i_0}(v), \ell, N) \in M_{i_0}(v')$. Suppose that an element n_1 has been added to $N(v)$ at a step $i_1 \in [i_0, i]$. The rule $\mathcal{M}_3$ or $\mathcal{M}_4$ has been applied and therefore $n_1 \leq n_0$, since $n_0 \in N_{i_1}(v)$. Consequently, $m_{i_0}(v) = m_i(v)$. Since $n_i(v') \neq n_{i_0}(v')$, the rule $\mathcal{M}_2$ or $\mathcal{M}_3$ has been applied to the node v' at a step $i_2 > i_0$ and therefore $n_i(v') \geq n_{i_2}(v') > m_{i_0}(v) = m_i(v)$ since there exists $(m_{i_0}(v), \ell, N) \in M_{i_0}(v') \subseteq M_{i_2-1}(v')$. Consequently, the property is also satisfied. □

Since we can ensure that the algorithm always terminates, we can give some properties of the final labelling:

Lemma 4.6. Any execution ρ of the enumeration algorithm on a connected labelled graph $\mathbf{G} = (G, \lambda)$ terminates and yields to a final labelling $(\lambda, n_\rho, N_\rho, M_\rho)$ satisfying the following conditions:

1. there exists an integer $k \leq |V(G)|$ such that $\{n_\rho(v) \mid v \in V(G)\} = [1, k]$,

and for all vertices v, v':

2. $M_\rho(v) = M_\rho(v')$,
3. $(n_\rho(v), \lambda(v), N_\rho(v)) \in M_\rho(v')$,
4. $n_\rho(v) = n_\rho(v')$ implies that $\lambda(v) = \lambda(v')$ and $N_\rho(v) = N_\rho(v')$,
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, $n_\rho(v) \in N_\rho(w)$.

Proof:

1. By Lemma 4.3 and Lemma 4.4 applied to the final labelling and since the rule $\mathcal{M}_2$ cannot be applied.
2. Otherwise, the rule $\mathcal{M}_1$ could be applied.
3. A direct corollary of the previous property using Lemma 4.1.
4. Otherwise, the rule $\mathcal{M}_2$ could be applied to v or v'.
5. By Lemma 4.5 and since no relabelling step can be performed anymore.

□

We can therefore prove that there exists a graph $\mathbf{H}$ associated to the final labelling of $\mathbf{G}$ such that $\mathbf{G}$ is a submersion of $\mathbf{H}$.

Proposition 4.2. Given a graph $\mathbf{G}$, we can associate with the final labelling of any execution ρ of the enumeration algorithm on $\mathbf{G}$, a graph $\mathbf{H}$ such that there exists a locally surjective homomorphism from $\mathbf{G}$ onto $\mathbf{H}$.

Proof:
We use the notation of Lemma 4.6. Let $\mathbf{G} = (G, \lambda)$.

Consider the graph H defined by $V(H) = \{m \in \mathbb{N} \mid \exists v \in V(G), n_\rho(v) = m\}$ and $E(H) = \{\{m, m'\} \mid \exists v, v' \in V(G); n_\rho(v) = m, n_\rho(v') = m'$ and $\{v, v'\} \in E(G)\}$.

From Lemma 4.6, $\{m, m'\} \in E(H)$ if and only if there exist $v, v' \in V(G)$ such that $n_\rho(v) = m, n_\rho(v') = m'$, $m' \in N_\rho(v)$ and $m \in N_\rho(v')$. From Lemma 4.1, we can conclude that there does not exist any $\{n, n\} \in E(H)$: the graph H does not contain self-loop. From the definition of $E(H)$, we deduce that H does not contain multiple edges.

From Lemma 4.6, $m \in N_H(n_\rho(v))$ if and only if there exists $w \in N_G(v)$ such that $n_\rho(w) = m$ and therefore n_ρ is a locally surjective homomorphism of unlabelled graphs from G onto H.

It remains to define the labelling λ_H on H. This is natural, just set $\lambda_H(n) = \lambda(v)$ for $v \in n_\rho^{-1}(n)$. From Lemma 4.6, if two nodes v, v' are such that $n_\rho(v) = n_\rho(v')$, then $\lambda(v) = \lambda(v')$. Consequently, this labelling is well-defined and obviously $\mathbf{G}$ is a submersion of $\mathbf{H} = (H, \lambda_H)$ via the homomorphism n_ρ.
□

Consider a graph $\mathbf{G}$ that is submersion-minimal. For every run ρ of the enumeration algorithm, the graph associated to the final labelling is isomorphic to $\mathbf{G}$ and therefore the set of numbers of the vertices is exactly $\{1, \ldots, |V(G)|\}$. The termination detection of the algorithm is possible on $\mathbf{G}$. Indeed, once a vertex gets the identity number $|V(G)|$, from Lemma 4.3 and Lemma 4.4, it knows that all the vertices have different identity numbers that will not change any more and it can conclude that the computation is over.

Furthermore, it has been shown in Lemma 4.1 that for every graph $\mathbf{G}$ that is not submersion-minimal, there exists no algorithm using cellular edge local computations to solve the naming problem or the enumeration problem on $\mathbf{G}$. Thus we have proven the following theorem.

Theorem 4.1. For every graph $\mathbf{G}$, the following statements are equivalent:

1. there exists a naming algorithm on $\mathbf{G}$ using cellular edge local computations,
2. there exists a naming algorithm with termination detection on $\mathbf{G}$ using cellular edge local computations,
3. there exists an enumeration algorithm on $\mathbf{G}$ using cellular edge local computations,
4. there exists an enumeration algorithm with termination detection on $\mathbf{G}$ using cellular edge local computations,
5. the graph $\mathbf{G}$ is a submersion-minimal graph.

5. Election Problem

If we can solve the enumeration problem on $\mathbf{G}$ then we can solve the election problem on $\mathbf{G}$; 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 homomorphism 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

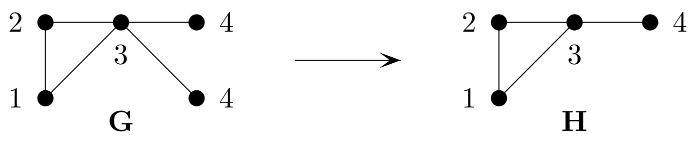


Figure 5. A graph for which we can solve the election problem but not the enumeration problem.

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 it can declare itself as elected.

5.1. Necessary Conditions to Solve the Election Problem

We would like to give here necessary conditions characterizing the graphs that admits an election algorithm. 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 3.1, 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 $\mathbf{H} \in \mathcal{S}_{\mathbf{G}}$.

Remark 5.1. 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 homomorphism 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 let us set $C_{\mathbf{H}}(\mathbf{G}) = V(H) \setminus P_{\mathbf{H}}(\mathbf{G})$ (the elements of this set are called the candidates of $\mathbf{H}$ for $\mathbf{G}$). From Remark 5.1, 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, in this case there exists an execution of the algorithm on $\mathbf{G}$ such that the label *elected* appears once in $\mathbf{G}_1$ and once in $\mathbf{G}_2$, contradicting the election principle. Recapitulating:

Proposition 5.1. 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. $\mathbf{H}_2 \in \mathcal{S}_{\mathbf{G}}$).

5.2. Election Algorithm

We now consider a graph $\mathbf{G}$ satisfying the conditions of Proposition 5.1.

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 [17].

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

5.2.1. The SSP Algorithm

This algorithm was originally devised to detect the termination of an algorithm in the message passing model. We consider a distributed algorithm which terminates when all processes reach their local termination conditions. Each process is able to determine only its own termination condition. The SSP algorithm detects an instant in which the entire computation is achieved.

Let G be a graph, to each node v is associated a predicate $P(v)$ and an integer $a(v)$, its confidence level. Initially $P(v)$ is false and $a(v)$ is equal to -1. If a vertex v has finished its computation of the initial algorithm, then it changes its value $P(v)$ to true. Each time a vertex changes the value of $P(v)$ or $a(v)$ then it informs its neighbours.

Transformations of the value of $a(v)$ are defined by the following rules. Each computation step acts on the integer $a(v_0)$ associated to the vertex v_0; the new value of $a(v_0)$ depends on the information v_0 have about the values $\{a(v_1), \ldots, a(v_d)\}$ of its neighbours. More precisely,

- If $P(v_0) = false$ then $a(v_0) = -1$;
- if $P(v_0) = true$ then $a(v_0) = 1 + Min\{a(v_k) \mid 0 \leq k \leq d\}$.

We will adapt this algorithm using the ideas of the algorithm GSPP [10]. For every vertex v, the value of $P(v)$, instead of being boolean, will be a graph reconstructed from the contents of the mailbox of v. An important property of the function P is that it is constant between two moments where it has the same value.

In our model, a vertex cannot distinguish its neighbours: therefore, we will use the numbers that appear in the local view: a vertex v will increase its confidence level $a(v)$ only if for each number n in its local view, it has a neighbour v' such that $n(v') = n$ and $a(v') \geq a(v)$.

5.2.2. Labels

As in Section 4.2, 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 ordered pair $(n(v'), a(v'))$ to $N(v)$. Thus $N(v)$ is always a finite set of ordered pairs 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.

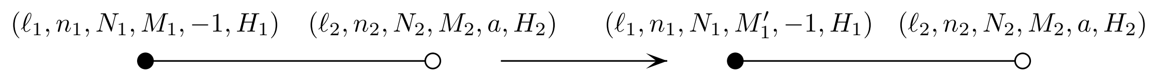
- $M(v) \subseteq \mathbb{N} \times L \times \mathcal{P}_{\text{fin}}(\mathbb{N})$ is the *mailbox* of v and contains the information received by v about the identity numbers existing in the graph and the local views associated with these numbers.

- $H(v)$ 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.

5.2.3. The Relabelling Rules

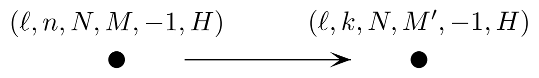
The first computation step $\mathcal{S}_0$ replaces just the initial label $\lambda(v)$ by $(\lambda(v), 0, \emptyset, \emptyset, -1, \emptyset)$. The following four rules mimic the rules of the enumeration algorithm:

$\mathcal{S}_1$:


$(\ell_1, n_1, N_1, M_1, -1, H_1)$
$(\ell_2, n_2, N_2, M_2, a, H_2)$
$(\ell_1, n_1, N_1, M_1', -1, H_1)$
$(\ell_2, n_2, N_2, M_2, a, H_2)$

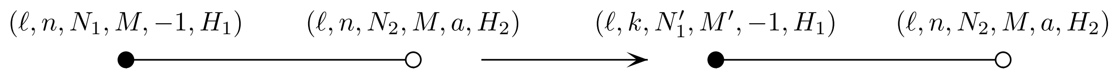
If $M_2 \setminus M_1 \neq \emptyset$ then $M_1' := M_1 \cup M_2$.

$\mathcal{S}_2$:



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
$M' := M \cup \{(k, \ell, \Pi_1(N))\}$.

$\mathcal{S}_3$:


$(\ell, n, N_1, M, -1, H_1)$
$(\ell, n, N_2, M, a, H_2)$
$(\ell, k, N_1', M', -1, H_1)$
$(\ell, n, N_2, M, a, H_2)$

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

$\mathcal{S}_4$:

$$\begin{tikzcd} \overset{(\ell_1, n_1, N_1, M, -1, H_1)}{\bullet} \arrow[r, dash] & \overset{(\ell_2, n_2, N_2, M, a, H_2)}{\circ} \arrow[r] & \overset{(\ell_1, n_1, N_1', M', -1, H_1)}{\bullet} \arrow[r, dash] & \overset{(\ell_2, n_2, N_2, M, a, H_2)}{\circ} \end{tikzcd}$$

If $n_1 > 0$, $n_2 > 0$, $n_1 \neq n_2$ and
$\forall (n_1, \ell_1', K_1') \in M$, $(\ell_1', K_1') \preceq (\ell_1, \Pi_1(N_1))$ and
$\forall (n_2, \ell_2', K_2') \in M$, $(\ell_2', K_2') \preceq (\ell_2, \Pi_1(N_2))$ and
$(n_2, -1) \notin N_1$
then $N_1' := N_1 \setminus \{(n', a') \in N_1 \mid n' < n_2\} \cup \{(n_2, -1)\}$ 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 contents 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 ordered pair (ℓ, N) for the order $\prec$. This operation gives a new mailbox contents that we shall denote by $u(M)$.

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 $n_2, \in N_1$ and $n_1 \notin N_2$ 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 $E(G_M) = \{\{n_1, n_2\} \mid \exists (n_1, \ell_1, N_1), (n_2, \ell_2, N_2) \in u(M), n_2 \in N_1$ and $n_1 \in N_2\}$. 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.

$\mathcal{S}_5$:

$$\begin{tikzcd} \overset{(\ell, n, N, M, a, H)}{\bullet} \arrow[r] & \overset{(\ell, n, N, M, a+1, H)}{\bullet} \end{tikzcd}$$

Applicable whenever $n > 0$ and
$\forall (n, \ell', K') \in M, (\ell', K') \preceq (\ell, \Pi_1(N))$ and
$\mathbf{G}_M \in \mathcal{S}_{\mathbf{G}}$ 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 neighbours if the confidence level of this neighbour has increased.

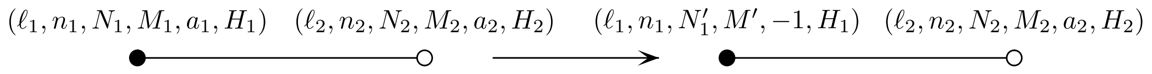
$\mathcal{S}_6$:

$$\begin{tikzcd} \overset{(\ell_1, n_1, N_1, M, a_1, H_1)}{\bullet} \arrow[r, dash] & \overset{(\ell_2, n_2, N_2, M, a_2, H_2)}{\circ} \arrow[r] & \overset{(\ell_1, n_1, N_1', M, a_1, H_1)}{\bullet} \arrow[r, dash] & \overset{(\ell_2, n_2, N_2, M, a_2, H_2)}{\circ} \end{tikzcd}$$

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

The rules $\mathcal{S}_7, \mathcal{S}_8, \mathcal{S}_9$ are designed to avoid deadlock in the execution of the algorithm. If one of these rules is applied on a vertex v, then the mailbox of v is modified and therefore, we have to reset its confidence level. 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 the contents of its mailbox, then it modifies also its history $H(v)$, so as to remember its former confidence level.

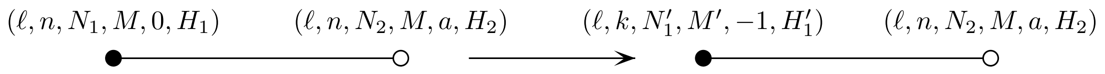
$\mathcal{S}_7$:


$(\ell_1, n_1, N_1, M_1, a_1, H_1)$
$(\ell_2, n_2, N_2, M_2, a_2, H_2)$
$(\ell_1, n_1, N_1', M', -1, H_1)$
$(\ell_2, n_2, N_2, M_2, a_2, H_2)$

If $M_2 \setminus M_1 \neq \emptyset$ 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
$H_1' := H_1 \cup \{(n_1, N_1, M_1, a_1)\}$.

The rule $\mathcal{S}_8$ enables a vertex v whose confidence level is equal to 0 to modify its state if it discovers the existence of a neighbour that has the same number as it. If a such vertex changes its state, then it modifies also its history $H(v)$, so as to remember its former confidence level.

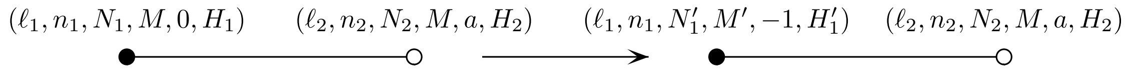
$\mathcal{S}_8$:


$(\ell, n, N_1, M, 0, H_1)$
$(\ell, n, N_2, M, a, H_2)$
$(\ell, k, N_1', M', -1, H_1')$
$(\ell, n, N_2, M, a, H_2)$

If $n > 0$ and
$\Pi_1(N_1) = \Pi_1(N_2)$ and
$\forall (n, \ell', K') \in M$, $(\ell', K') \preceq (\ell, \Pi_1(N_1))$
then $k := 1 + \max\{n' \mid \exists (n', \ell', K') \in M\}$,
$N_1' := \{(n', -1) \mid \exists (n', a') \in N_1; n' > n\} \cup \{(n, -1)\}$,
$M' := M \cup \{(k, \ell, \Pi_1(N_1'))\}$ and
$H_1' := H_1 \cup \{(n, N_1, M, 0)\}$.

The rule $\mathcal{S}_9$ enables a vertex v whose confidence level is equal to 0 to modify its state if it discovers the existence of a neighbour it didn't know, i.e., the number of its neighbour does not appear in its local view. If a such vertex changes its state, then it modifies also its history $H(v)$, so as to remember its former confidence level.

$\mathcal{S}_9$:

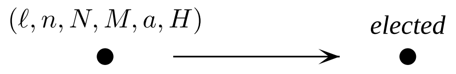

$(\ell_1, n_1, N_1, M, 0, H_1)$
$(\ell_2, n_2, N_2, M, a, H_2)$
$(\ell_1, n_1, N_1', M', -1, H_1')$
$(\ell_2, n_2, N_2, M, a, H_2)$

If $n_1 > 0$, $n_2 > 0$, $n_1 \neq n_2$ and
$\forall (n_1, \ell_1', K_1') \in M$, $(\ell_1', K_1') \preceq (\ell_1, \Pi_1(N_1))$ and
$\forall (n_2, \ell_2', K_2') \in M$, $(\ell_2', K_2') \preceq (\ell_2, \Pi_1(N_2))$ and
$\nexists (n_2, a') \in N_1$

then $N_1' := \{(n, -1) \mid \exists (n, a') \in N_1; n > n_2\} \cup \{(n_2, -1)\}$,
$M' := M \cup \{(n_1, \ell_1, N_1')\}$ and
$H_1' := H_1 \cup \{(n_1, N_1, M, 0)\}$.

The rule $\mathcal{S}_{10}$ enables a node that has a confidence level equal to $|V(G)| + 2$ to get the label *elected* if and only if its identity number is the greatest number belonging to the candidates of the reconstructed graph for $\mathbf{G}$.

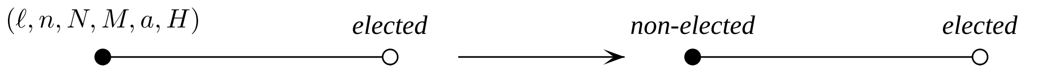
$\mathcal{S}_{10}$:



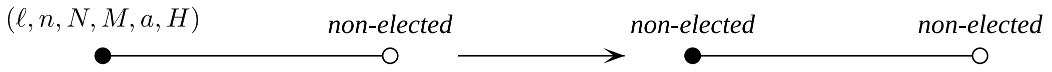
Applicable whenever $n = \max\{n \in C_{\mathbf{G}_M}(\mathbf{G})\}$ and
$a = |V(G)| + 2$.

The last rules enable to propagate the information, once a node got the final label *elected* it informs all the other nodes of the graphs that they are *non-elected*.

$\mathcal{S}_{11}$:



$\mathcal{S}_{12}$:



5.3. Correctness of the Election Algorithm

We will denote the algorithm described above by $\mathcal{S}$ and we will first consider the algorithm $\mathcal{S}'$ described by the rules $\mathcal{S}_1, \ldots, \mathcal{S}_9$. If we can ensure the termination of $\mathcal{S}'$, we can prove that $\mathcal{S}$ always terminates, since the rules $\mathcal{S}_{10}, \mathcal{S}_{11}, \mathcal{S}_{12}$ can modify the label of each vertex only once. Clearly the labels *elected* and *non-elected* are terminal. Moreover, if we show that exactly one vertex can apply the rule $\mathcal{S}_{10}$ then it will prove the correctness of the algorithm. Indeed, there will be exactly one vertex with the label *elected* in the final configuration whereas all the other vertices will have the label *non-elected*.

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 algorithm $\mathcal{S}'$. As for the precedent algorithm, we can show that the algorithm has interesting monotonicity properties.

Lemma 5.1. For each step i, for each vertex v,

1. $n_i(v) \leq n_{i+1}(v)$,
2. $M_i(v) \subseteq M_{i+1}(v)$,
3. $H_i(v) \subseteq H_{i+1}(v)$,
4. if $H_i(v) = H_{i+1}(v)$ then $\Pi_1(N_i(v)) \preceq \Pi_1(N_{i+1}(v))$, $a_i(v) \leq a_{i+1}(v)$ and $\forall (n, a_1) \in N_i(v)$, $(n, a_2) \in N_{i+1}(v), a_1 \leq a_2$.

And for each step i, there exists at least one vertex v, such that one of the inequalities is strict for v.

As in the preceding algorithm, we can show that for every vertex v and for every step i, the value of $n_i(v)$ is always lower or equal to $|V(G)|$, and since the value of $a_i(v)$ is bounded by $|V(G)| + 2$, we can easily deduce that the values of $N_i(v), M_i(v)$ and $H_i(v)$ are also bounded. From Lemma 5.1, we can therefore conclude that each execution of the algorithm $\mathcal{S}'$ terminates.

In the following lemma, we prove that if at a step i, a vertex v knows the confidence level a of one of its neighbours then $a \geq a_i(v) - 1$,

Lemma 5.2. For every vertex v and step i, for every $(n, a) \in N_i(v)$, $a \geq a_i(v) - 1$.

Proof:
In the initial configuration, this result is obviously true. Consider now a step i such that this property is true after this step. Suppose that at step $i + 1$, one of the rules $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4, \mathcal{S}_7, \mathcal{S}_8, \mathcal{S}_9$ is applied to a vertex v, then $a_{i+1}(v) = -1$ and the property holds.

If the rule $\mathcal{S}_5$ is applied at step $i + 1$, for all $(n, a) \in N_i(v) = N_{i+1}(v)$, $a \geq a_i(v) = a_{i+1}(v) - 1$: the property holds.

If the rule $\mathcal{S}_6$ is applied to v according to the label of one of its neighbours v' then $(n_i(v'), a_1)$ is replaced by $(n_i(v'), a_2)$ in $N_i(v)$ to obtain $N_{i+1}(v)$. Consequently, $a_{i+1}(v) = a_i(v)$ and $a_2 > a_1 \geq a_{i+1}(v) - 1$, the property holds again. □

In the following lemma, we show that if the rule $\mathcal{S}_7$ is applied at a step $j + 1$ to a vertex v whose confidence level $a_j(v)$ was greater than 1 then there exists a neighbour v' of v such that the rule $\mathcal{S}_7$ was applied to v' at a step $j' + 1 \leq j$, where $M_j(v) = M_{j'}(v')$ and $a_{j'}(v') \geq a_j(v) - 1$.

Lemma 5.3. For every vertex $v \in V(G)$ and every step j such that $a_j(v) \geq k + 1 \geq 1$ and $M_j(v) \subsetneq M_{j+1}(v)$, there exists a vertex $v' \in N_G(v)$ and a step $j' < j$ such that $M_{j'}(v') = M_j(v) \subsetneq M_{j'+1}(v')$ and $a_{j'}(v') \geq k$.

Proof:
Since $a_j(v) \geq k + 1$ and $M_j(v) \subsetneq M_{j+1}(v)$, it means that $a_{j+1}(v) = -1$ and that the rule $\mathcal{S}_7$ has been applied to v according to the label of one of its neighbours v' at the step $j + 1$.

Thus there exists $(n, N, M_j(v), a) \in H_j(v')$ such that $a \geq a_j(v) - 1 \geq k$. If $a \geq 1$, then there exists a step $j' + 1 \leq j$ such that the rule $\mathcal{S}_7$ has been applied to v' at this step and therefore $a_{j'}(v') = a \geq k$ and $M_{j'}(v') = M_j(v) \subsetneq M_{j'+1}(v')$.

If $a = 0$, then $k = 0$ and there exists a step $j' + 1$ such that one of the rules $\mathcal{S}_7, \mathcal{S}_8, \mathcal{S}_9$ has been applied to v' at this step. Consequently, $a_{j'}(v') = 0 \geq k$ and $M_{j'}(v') = M_j(v) \subsetneq M_{j'+1}(v')$. □

In the following proposition, we prove that each execution of the algorithm $\mathcal{S}'$ on a graph $\mathbf{G}$ does not stop before at least one vertex gets a confidence level equals to $|V(G)| + 2$.

Proposition 5.2. For each execution of $\mathcal{S}'$ on $\mathbf{G}$, there exists a vertex $v \in V(G)$ and a step i such that $a_i(v) = |V(G)| + 2$.

Proof:
We already know that each execution of $\mathcal{S}'$ terminates. Consider an execution ρ of $\mathcal{S}'$ that yields to a final labelling $(\lambda, n_\rho, N_\rho, M_\rho, a_\rho, H_\rho)$. Suppose that for each vertex v, $a_\rho(v) \leq |V(G)| + 1$.

Suppose that for each vertex v, $a_\rho(v) = -1$, then using the results of the preceding section, the final numbering n_ρ induces a graph $\mathbf{H}$ such that $\mathbf{G}$ is a submersion of $\mathbf{H}$ via n_ρ, i.e., $\mathbf{H} \in \mathcal{S}_\mathbf{G}$. Consequently, for each vertex v, $\mathbf{G}_{M_\rho(v)} = \mathbf{H} \in \mathcal{S}_\mathbf{G}$, and since the rule $\mathcal{S}_2$ cannot be applied on v, one can apply the rule $\mathcal{S}_5$ on v: the computation is not finished.

Consider now a vertex v such that $a_\rho(v) \geq 1$. Since we cannot apply the rule $\mathcal{S}_5$ on v, from Lemma 5.2, there exists some $(n', a_\rho(v) - 1) \in N_\rho(v)$. Consider the last step j where the rule $\mathcal{S}_6$ has been applied by v according to the label of a neighbour v' such that $n_j(v') = n'$. Then $a_j(v') = a_\rho(v) - 1$, $M_\rho(v) = M_j(v) = M_j(v')$ and if $M_j(v') \subsetneq M_\rho(v')$, then the vertex v can apply the rule $\mathcal{S}_7$ according to the label of v'. Consequently, $M_j(v') = M_\rho(v')$, $n_j(v') = n_\rho(v')$ and $\Pi_1(N_j(v')) = \Pi_1(N_\rho(v'))$. Since we cannot apply the rule $\mathcal{S}_6$ on v according to the label of v', it implies that $a_\rho(v') = a_\rho(v) - 1$. Consequently, for each v such that $a_\rho(v) \geq 1$, there exists some vertex v' such that $M_\rho(v) = M_\rho(v')$ and $a_\rho(v') = 0$.

Consider now a vertex v such that $a_\rho(v) = 0$ and such that for each vertex v', if $a_\rho(v') \geq 0$, then either $M_\rho(v) = M_\rho(v')$ or $M_\rho(v') \setminus M_\rho(v) \neq \emptyset$. Since we cannot apply the rule $\mathcal{S}_5$ on v, from Lemma 5.2, there exists some $(n', -1) \in N_\rho(v)$. Consider the last step j where the rule $\mathcal{S}_6$ has been applied by v according to the label of a neighbour v' such that $n_j(v') = n'$. Then $a_j(v') = -1$.

If $a_\rho(v') = -1$, then since the rule $\mathcal{S}_1$ and $\mathcal{S}_7$ cannot be applied on v' or v, $M_\rho(v) = M_\rho(v')$ and $\mathbf{G}_{M_\rho(v')} \in \mathcal{S}_\mathbf{G}$. Since the rule $\mathcal{S}_5$ cannot be applied on v', it means that there exists $(n_\rho(v'), \ell, K) \in M_\rho(v')$ such that $(\lambda(v'), N_\rho(v')) \prec (\ell, K)$ and therefore, the rule $\mathcal{S}_2$ can be applied: the computation is not finished.

If $a_\rho(v') \geq 0$, then since the rule $\mathcal{S}_7$ cannot be applied on v according to the state of v', $M_\rho(v) = M_\rho(v')$. Suppose that $n_\rho(v) = n_\rho(v')$, then $\Pi_1(N_\rho(v)) = \Pi_1(N_\rho(v'))$ and therefore, one can apply the rule $\mathcal{S}_8$ on v according to the label of v'. Consequently, $n_\rho(v) \neq n_\rho(v')$ and from Lemma 4.5, either $n_\rho(v') = n_j(v')$ or $n_\rho(v') \notin \Pi_1(N_\rho(v))$. In the first case, one can apply the rule $\mathcal{S}_6$ on v according to the label of v'; in the second case, one can apply the rule $\mathcal{S}_9$ on v according to the label of v'.

Consequently, there does not exists any execution ρ of $\mathcal{S}'$ that terminates, such that in the final state, for each vertex $v \in V(G)$, $a_\rho(v) \leq |V(G)| + 1$. □

The most important property of the algorithm is given in the following proposition. Roughly speaking it states that if the confidence level of a vertex v is $|V(G)| + 2$ then $\mathbf{G}$ contains a submersion of $\mathbf{G}_{M(v)}$.

Proposition 5.3. If there exists a vertex $v_0 \in V(G)$ and a step i_0 such that $a_{i_0}(v_0) = |V(G)| + 2$ then $\mathbf{G}$ 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 $v \in V(\mathbf{H})$, $M_i(v) = M_{i_0}(v_0)$. Moreover, there exists a step $i_1 \geq i_0$ such that the label *elected* appears on one vertex $v \in V(H)$.

Proof:
Consider a vertex $v_0 \in V(G)$ and a step i_0 such that $a_{i_0}(v_0) = |V(G)| + 2$; we will consider $M_0 = M_{i_0}(v_0)$. We will define recursively a sequence of sets of vertices $(V_j)_{j\in\mathbb{N}}$ and we will need a partial function i over $V(G)$ defined as follows: given a vertex $v \in V(G)$, if there exists a step $j \leq i_0$ such that $M_j(v) = M_0$, then $i(v) = \max\{j \leq i_0 \mid M_j(v) = M_0\}$, otherwise $i(v)$ is not defined.

Let $V_0 = \{v_0\}$ and $V_{k+1} = V_k \cup \{w \mid \exists v \in N_G(w) \cap V_k; a_{i(w)}(w) \geq a_{i(v)}(v) - 1\}$.

Let $\mathbf{H} = (H, \nu)$ be the subgraph of $\mathbf{G}$ defined by $V(H) = \bigcup_{j\in\mathbb{N}} V_j = V_{|V(G)|}$ and $E(H) = \big\{\{v, v'\} \in E(G) \mid v, v' \in V(H); \exists (n_{i(v)}(v), a) \in N_{i(v')}(v'); a_{i(v)}(v) \geq a\big\}$. We define the labelling of $\mathbf{H}$ as follows: for every $v \in V(H)$, $\nu(v) = \lambda(v)$. Let γ the homomorphism from $\mathbf{H}$ onto $\mathbf{G}_{M_0}$ defined by $\gamma(v) = n_{i(v)}(v)$.

For each vertex $v \in V_p \setminus V_{p-1}$, there exists a simple path of length p from v to v_0. Therefore, for each vertex $v \in V(H)$ such that $a_{i(v)}(v) = |V(G)| + 2 - p$, there exists a simple path from v to v_0 of length greater or equal to p, since such a vertex belongs to $V_k \setminus V_{k-1}$ with $k \geq p$. Moreover, for every $v \in V(H)$, $a_{i(v)}(v) \geq |V(G)| + 2 - |V(H)| \geq 2$.

Consider a vertex $v \in V(H)$ such that $a_{i(v)}(v) \geq 1$. If $(n, a) \in N_{i(v)}(v)$ then it means that $a \geq a_{i(v)}(v) - 1 \geq 0$ and therefore the rule $\mathcal{S}_6$ has been applied at a step $j \leq i(v)$ between v and one of its neighbours w such that $n_j(w) = n$, $M_j(w) = M_{i(v)}(v) = M_0$ and $a_j(w) = a$. Consequently, $i(w)$ is defined and $a_{i(w)}(w) \geq a_j(w) = a \geq a_{i(v)}(v) - 1$. Therefore, $w \in V(H)$ and the homomorphism γ is locally surjective for every vertex $v \in V(H)$ such that $a_{i(v)}(v) \geq 1$. Since for every vertex $v \in V(H)$, $a_{i(v)}(v) \geq 1$, the graph $\mathbf{H}$ is a submersion of $\mathbf{G}_{M_0}$.

Suppose that there exists a vertex $v \in V(H)$ and a step j such that $M_j(v) = M_0 \subsetneq M_{j+1}(v)$. Consider a vertex $v_1 \in V(H)$ and a step j_1 such that $M_{j_1}(v_1) = M_0 \subsetneq M_{j_1+1}(v_1)$ and for all $v \in V(H)$, there exists $j \geq j_1$ such that $M_j(v) = M_0(v)$.

Since $a_{j_1}(v_1) \geq |V(G)| + 2 - |V(H)|$, we can apply Lemma 5.3 to find a sequence of vertices $(v_1, \ldots, v_k)$ and a decreasing sequence of steps $(j_1, \ldots, j_k)$ such that $k = |V(G)| - |V(H)| + 1$ and for every $1 \leq p \leq k$, $M_{j_p}(v_p) = M_0 \subsetneq M_{j_p+1}(v_p)$ and $a_{j_p}(v_p) \geq k + 1 - p$. From the definition of v_1 we know that for every $p \geq 2$, $v_p \notin V(H)$. Thus, since $k = |V(G)| - |V(H)| + 1$, there exists $p > q$ such that $v_p = v_q$. But it implies that $M_0 = M_{j_p}(v_p) \subsetneq M_{j_p+1}(v_p) \subseteq M_{j_q}(v_q) = M_0$, which is impossible. Then, for every step $i \geq i_0$ for every vertex $v \in V(H)$, $M_i(v) = M_0$.

We will now show that the rule $\mathcal{S}_{10}$ can be applied on a vertex $v \in V(H)$ at a step $i_1 \geq i_0$. Clearly, if for each $v \in V(H)$, $a(v) = |V(G)| + 2$, then there exists one vertex $v \in V(H)$ such that $n(v) = \max\{n \in C_{\mathbf{G}_M}(\mathbf{G})\}$ and we can apply the rule $\mathcal{S}_{10}$ on this vertex.

Suppose now that for each step $i > i_0$, there exists some $v \in V(H)$ such that $a(v) < |V(G)| + 2$. Since the execution of $\mathcal{S}'$ terminates on $\mathbf{G}$, we will now consider the final labelling $(\lambda, n_\rho, N_\rho, M_\rho, a_\rho, H_\rho)$. In the final configuration, $M_\rho(v_0) = M_{i_0}(v_0)$, and $a_\rho(v_0) = a_{i_0}(v_0) = |V(G)| + 2$. We can therefore redefine the graph $\mathbf{H}$ as before by considering the final computation step instead of i_0.

Consider now a vertex $v \in V(H)$ such that for each vertex $v' \in V(H)$, $a_\rho(v) \leq a_\rho(v')$. Since the rule $\mathcal{S}_5$ cannot be applied on v, it implies that there exists some $(n, a_\rho(v) - 1) \in N_\rho(v)$. Consider the last step j such that the rule $\mathcal{S}_6$ has been applied between v and a vertex v' such that $n_j(v') = n$. As explained before, $v' \in V(H)$ and $\{v, v'\} \in E(H)$. Consequently, $a_j(v') = a_\rho(v) - 1$ and $M_\rho(v) = M_j(v) = M_j(v') = M_\rho(v')$. Therefore $n_\rho(v') = n_j(v')$ and $a_\rho(v') \geq a_\rho(v) > a_\rho(v) - 1$. The rule $\mathcal{S}_6$ can then be applied by v according to the label of v': the computation is not terminated.

This implies that there exists a step $i_1 > i_0$ such that one can apply the rule $\mathcal{S}_{10}$ on some vertex and this vertex gets the label *elected*.

□

From Proposition 5.3, if a vertex v_0 has a level of confidence greater than $|V(G)| + 2$ at a step i_0, there exists a subgraph $\mathbf{H}$ of $\mathbf{G}$ that is a submersion of $\mathbf{G}_{M_{i_0}(v_0)} \in \mathcal{S}_\mathbf{G}$ such that for every $v \in V(H)$ and for every step $i > i_0$, $\mathbf{G}_{M_i(v)} = \mathbf{G}_{M_{i_0}(v_0)}$ and $a_i(v) \geq 1$.

We know that 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. $\mathbf{H}_2 \in \mathcal{S}_\mathbf{G}$). And so we can choose to elect one candidate in $\mathbf{H}$, which is possible as $C_\mathbf{H}(\mathbf{G}) \neq \emptyset$: there will be at most one vertex that will take the label *elected* by the rule $\mathcal{S}_{10}$ and therefore the algorithm $\mathcal{S}$ is an election algorithm for $\mathbf{G}$.

Consequently, for each graph $\mathbf{G}$ that satisfies the necessary conditions of Proposition 5.1 there exists an algorithm that solves the election problem over $\mathbf{G}$:

Theorem 5.1. 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. $\mathbf{H}_2 \in \mathcal{S}_\mathbf{G}$).

6. Examples

If we assume that each vertex of a graph $\mathbf{G}$ has a unique identifier then $\mathbf{G}$ is a submersion-minimal and the knowledge of its size allows the election.

6.1. Trees, Grids and Complete graphs

Consider a graph G with at least three vertices that can be coloured with two colours. Such a colouring yields a submersion of G onto the graph K_2 with two vertices and one edge between them. Such a submersion is non trivial if G has at least three vertices and therefore there does not exist a naming algorithm using cellular edge local computations for G.

Moreover, if $\varphi_1 : G \to K_2$ is a submersion (colouring) of $\mathbf{G}$ then exchanging the two colours we get another submersion φ_2 and if G has at least three vertices then for each vertex $k \in V(K_2)$ at least one of the sets $\varphi_i^{-1}(k)$, $i = 1, 2$, has cardinality ≥ 2. Thus, the election problem cannot be solved with cellular edge local computations for graphs with at least three vertices that admit a 2-coloring.

In particular, in our model, we cannot solve either naming or election for trees or grids that have at least three vertices, as they can be coloured with only two colors.

But complete graphs, since they are submersion-minimal, admit both naming and election algorithms in our model.

6.2. Rings with a Prime Size

It is well known that it is impossible to solve the naming and the election problems on a ring in the asynchronous message passing model [18]. And consequently to solve these problems on rings, we need synchronization. Unfortunately, even in the powerful model of Mazurkiewicz, we cannot solve these problems if the size of the ring is not a prime number (except for the ring with four vertices). In the following we will show that in our model, even if it is much more weaker than Mazurkiewicz's model, we can still solve both problems for the rings of prime size.

First note the following fact:

Proposition 6.1. An unlabelled ring of size p is submersion-minimal if and only if p is prime.

Proof:
Consider a ring R_p of size p: $V(R_p) = \{0, 1, \ldots, p-1\}$, $E(R_p) = \{(i, i+1 \mod p) \mid 0 \leq i < p\}$.

If u divides p then the mapping $\varphi(i) = i \mod u$ is obviously a submersion from R_p onto the ring R_u of size u. To be more precise, for the particular case of $u = 2$, R_2 will denote here the graph with two vertices $0, 1$ and an edge $\{0, 1\}$ rather than a ring.

On the other hand, suppose now that $\varphi : R \to H$ is a non trivial submersion. Since the image of any vertex of degree 2 under a submersion has either degree 1 or 2 the graph H is either a ring or a chain C_l (C_l has l vertices $V(C_l) = \{c_0, \ldots, c_{l-1}\}$ and $l-1$ edges $E(C_l) = \{(c_i, c_{i+1}) \mid 0 \leq i < l-1\}$). In the first case the size of H divides the size of R_p and the corresponding submersion is essentially the one described already in the first part of the proof. In the second case φ maps one vertex, say v, of R_p into c_0 and next the two vertices at the distance i from v are mapped to c_i, $1 \leq i < p/2$. At the distance $p/2$ from v there can be only one vertex in R_p that is mapped to c_{l-1}. Therefore in this case p is a multiple of 2. □

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 synchronization power. Moreover, contrary to some other algorithms on rings, our enumeration algorithm does not need any sense of direction.

7. Conclusion

In this work, we have characterized the graphs where we can solve the naming and the election problems. As for message passing systems, unique identities and the knowledge of the size of the networks enable both election and naming in our model.

The characterization of graphs that admit a naming algorithm is a nice characterization of the same kind as other results existing for different models [3, 4, 5, 15]. For all these models, one can find a particular type of locally constrained homomorphisms (here we deal with submersions) such that the graphs where we can solve the naming problem are the graphs that are minimal according to this type of locally constrained homomorphisms.

Moreover, one can note that for every submersion-minimal graph, our algorithm $\mathcal{M}$ terminates. If the vertices know the size of the network, then they can detect the termination of the algorithm. We have a naming algorithm (without termination detection) for the class of submersion-minimal graphs, and a naming algorithm with termination detection for the class of submersion-minimal graphs of size n.

The characterization of graphs that admit an election algorithm is quite involved, and one can note that the processes have to know the graph $\mathbf{G}$ in order to compute $\mathcal{S}_{\mathbf{G}}$: for two different graphs, one need a different initial knowledge. In the model of Mazurkiewicz, there exists a characterization of classes of graph that admits an election algorithm [11, 12], an interesting problem is to find such a characterization in our model.

In Section 6, we have proved that there is no election algorithm for trees or grids with cellular edge local computations. A natural problem is to know what minimal initial knowledge enables election for these graphs. More generally, another interesting problem is to compare the power of cellular edge local computations with initial knowledge (the degree of the vertices, the size of the network, a bound on the size of the network, ...) with the different models of local computations.

References

[1] Angluin, D.: Local and global properties in networks of processors, *Proceedings of the 12th Symposium on Theory of Computing*, 1980.

[2] Attiya, H., Welch, J.: *Distributed computing: fundamentals, simulations, and advanced topics*, McGraw-Hill, 1998.

[3] Boldi, P., Codenotti, B., Gemmell, P., Shammah, S., Simon, J., Vigna, S.: Symmetry breaking in anonymous networks: Characterizations, *Proc. 4th Israeli Symposium on Theory of Computing and Systems*, IEEE Press, 1996.

[4] Chalopin, J.: Election and local computations on closed unlabelled edges (*extended abstract*), *Proc.of SOFSEM 2005*, number 3381 in LNCS, 2005.

[5] Chalopin, J., Métivier, Y.: Election and local computations on edges (*ext. abstract*), *Proc. of Foundations of Software Science and Computation Structures, FOSSACS'04*, number 2987 in LNCS, 2004.

[6] Chalopin, J., Métivier, Y.: A bridge between the asynchronous message passing model and local computations in graphs (*ext. abstract*), *Proc. of Mathematical Foundations of Computer Science, MFCS'05*, number 3618 in LNCS, 2005.

[7] Chalopin, J., Métivier, Y., Zielonka, W.: Election, naming and cellular edge local computations (*ext. abstract*), *Proc. of International conference on graph transformation, ICGT'04*, number 3256 in LNCS, 2004.

[8] Fiala, J., Paulusma, D.: A complete complexity classification of the role assignement problem, *Theoretical computer science*, to appear.

[9] Fiala, J., Paulusma, D., Telle, J. A.: Matrix and graph orders derived from locally constrained graph homomorphisms, *Proc. of Mathematical Foundations of Computer Science, MFCS'05*, number 3618 in LNCS, 2005.

[10] Godard, E., Métivier, Y., Muscholl, A.: Characterization of classes of graphs recognizable by local computations, *Theory of Computing Systems*, **37**(2), 2004, 249–293.

[11] Godard, E., Métivier, Y., Tel, G.: Election, termination and graph cartography, in preparation.

[12] Godard, E., Métivier, Y.: A characterization of families of graphs in which election is possible (*ext. abstract*), *Proc. of Foundations of Software Science and Computation Structures, FOSSACS'02*, number 2303 in LNCS, Springer-Verlag, 2002.

[13] LeLann, G.: Distributed systems: Towards a formal approach, *Information processing'77* (B. Gilchrist, Ed.), North-Holland, 1977.

[14] Lynch, N. A.: *Distributed algorithms*, Morgan Kaufman, 1996.

[15] Mazurkiewicz, A.: Distributed enumeration, *Inf. Processing Letters*, **61**(5), 1997, 233–239.

[16] Mazurkiewicz, A.: Bilateral ranking negotiations, *Fundamenta Informaticae*, **60**, 2004, 1–16.

[17] Szymanski, B., Shy, Y., Prywes, N.: Terminating iterative solutions of simultaneous equations in distributed message passing systems, *Proc. of the 4th Symposium on Principles of Distributed Computing, PODC'85*, 1985.

[18] Tel, G.: *Introduction to distributed algorithms*, Cambridge University Press, 2000.

[19] Yamashita, M., Kameda, T.: Computing on anonymous networks: Part I - Characterizing the solvable cases, *IEEE Transactions on parallel and distributed systems*, **7**(1), 1996, 69–89.

[20] Yamashita, M., Kameda, T.: Leader election problem on networks in which processor identity numbers are not distinct, *IEEE Transactions on parallel and distributed systems*, **10**(9), 1999, 878–887.