Fundamenta Informaticae 72 (2006) 1–26
IOS Press

An Efficient Message Passing Election Algorithm based on Mazurkiewicz's Algorithm*

Jérémie Chalopin, Yves Métivier†
LaBRI
Université de Bordeaux
{chalopin,metivier}@labri.fr

Abstract. We study the election and the naming problems in the asynchronous message passing model. We present a necessary condition based on Angluin's lifting lemma [Ang80] that must be satisfied by any network that admits a naming (or an election) algorithm. We then show that this necessary condition is also sufficient: we present an election and naming algorithm based on Mazurkiewicz's algorithm [Maz97]. The algorithm we obtained is totally asynchronous and it needs a polynomial number of messages of polynomial size, whereas previous election algorithms in this model are pseudo-synchronous and use messages of exponential size.

Keywords: Distributed Algorithm, Asynchronous Message Passing Model, Anonymous Network, Local Computation, Election, Naming, Covering

1. Introduction

The understanding of properties of a network which enable to solve in a distributed way typical problems of distributed computing enhances our understanding of what can be computed in a distributed way. Such problems are election, naming, spanning tree construction, termination detection, network topology recognition, consensus, mutual exclusion, etc. Not only solutions to these problems constitute primitive building blocks for many other distributed algorithms, but these solutions generally rely on combinatorial tools that enable a more general study of what can be computed in a distributed way.

We consider networks with arbitrary topology that are represented by simple connected graphs endowed with a port-numbering that enables each process to distinguish its neighbours. In one computation

*A preliminary version of this paper has been published in [CM05].
†Address for correspondence: LaBRI, Université de Bordeaux, 351 cours de la Libération, 33405 Talence, France

step, each process can either modify its state, or send a message to one of its neighbours, or receive a message from a neighbour. We consider the asynchronous message passing model: processes cannot access a global clock, processes execute computation steps at arbitrary speed, and a message sent from a process to a neighbour arrives within some finite but unpredictable time.

1.1. Election and Naming

In this paper, we focus on two classical problems of distributed computing that are election and naming. The election problem is one of the paradigms of the theory of distributed computing. It was first posed by LeLann [LeL77]. A distributed algorithm solves the election problem if it always terminates and in the final configuration exactly one process is marked as *elected* and all the other processes are *non-elected*. Moreover, it is supposed that once a process becomes *elected* or *non-elected* then it remains in such a state until the end of the execution of the algorithm. Election algorithms constitute a building block of many other distributed algorithms. The elected vertex acts as coordinator, initiator, and more generally performs some special role [TvS02]. If processes have initially unique identifiers, it is always possible to solve this problem by electing the process with the smallest identifier. Nevertheless, if we consider *anonymous* networks where processes do not have identifiers and execute the same algorithm, it is not always possible to solve the election problem. One aim of this paper is to present a characterization of networks where this problem can be solved.

The naming problem is another important problem in the theory of distributed computing. The aim of a naming algorithm is to arrive at a final configuration where all processes have unique identities (we assume that identities are totally ordered). Many distributed algorithms work correctly only under the assumption that all processes can be unambiguously identified thus it is very important to be able to give dynamically and in a distributed way unique identities to all processes.

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

1.2. Related Works

Since the pioneer work of Angluin [Ang80], it is well-known that there exists networks that do not admit any election algorithm since they are too "symmetric". The model of Angluin is defined in the following way. A network is a simple undirected connected graph with a port-numbering. A process is attached to each vertex and there is no global time (the network is asynchronous). A basic computation step is a pairwise exchange of messages by the two processes at the two ends of some edge. This exchange allows the two processes to change their internal states and does not affect any other process. To break the symmetry between two adjacent processes a "coin toss" may be used. In the Angluin's model, the combinatorial tool used to express these symmetries is the notion of *simple coverings*, i.e., locally bijective homomorphisms between simple graphs.

In [Maz97], Mazurkiewicz considers a model where in one computation step, a process can modify its state and the states of its neighbours by the application of some relabelling rule depending only on its previous state and the previous states of its neighbours. This model corresponds to a more abstract model of computation than the message passing model studied in this paper; it is a model where a computation step involves some synchronization between neighbouring processes. In his model, Mazurkiewicz shows

that a graph G admits an election (or a naming) algorithm if and only if G is not ambiguous (in [GMM04] it is proved that a graph G is non-ambiguous if and only if G is not a simple covering of another graph H distinct from G). In other words, in Mazurkiewicz's model, Angluin's necessary condition is also sufficient.

In the message passing model, Yamashita and Kameda give a characterization of networks admitting an election algorithm [YK96]. The characterization of Yamashita and Kameda is really different from Mazurkiewicz's result and the techniques they used are also different from the ones used by Mazurkiewicz. The characterization presented in [YK96] relies on the notion of "views", where the view of each vertex v in a graph G with a port-numbering ν is an infinite labelled rooted tree obtained by considering all labelled walks in G starting from v. Yamashita and Kameda first show that if two vertices v, v' of some network (G, ν) have the same view, there exists an execution of any algorithm such that v and v' always remain in the same state. Then they prove that if a network (G, ν) admits a naming (or an election) algorithm, all vertices must have distinct views. Yamashita and Kameda show that this necessary condition is also sufficient. Their algorithm relies on a result of Norris [Nor95] stating that two vertices of a network (G, ν) have the same view if and only if they have the same view up to height $|V(G)|$. Thus, each vertex just has to compute its view up to height $|V(G)|$ and to compare it with the view of the other vertices; the vertex with the "smallest" view is elected.

The results obtained by Boldi *et al.* [BCG$^+$96] enable to establish some links between the results of Mazurkiewicz and of Yamashita and Kameda. Boldi *et al.* consider a synchronous model of computation, but their results can be interpreted in the model studied by Yamashita and Kameda. The impossibility results presented in [BCG$^+$96] rely on some adaptation of the Angluin's lifting lemma and thus the characterizations obtained by Boldi *et al.* are expressed in terms of fibrations and coverings that are special homomorphisms between directed graphs. The characterization presented in [BCG$^+$96] is close to the existing characterization in Mazurkiewicz's model. Nevertheless, the algorithm presented by Boldi *et al.* to obtain sufficient conditions uses the same ideas as the algorithm of Yamashita and Kameda.

The algorithms of Yamashita and Kameda (and of Boldi *et al.*) are really different from the Mazurkiewicz'one. Indeed, Yamashita and Kameda's algorithm needs some initial knowledge on the network in order to enable each process to know what is the height of the view it should compute. On the other hand, Mazurkiewicz's algorithm does not need any initial knowledge to terminate, even if some initial knowledge is needed to enable the vertices to detect that the computation is over. This is an important property, since it is used by Godard et al. in [GMM04] to characterize graph classes that can be recognized in a distributed way in Mazurkiewicz's model. Moreover, to execute Mazurkiewicz's algorithm, it is sufficient for each vertex to have a memory of polynomial size, whereas in Yamashita and Kameda's algorithm, each process needs a memory of exponential size. This is an important property, since the size of messages in Yamashita and Kameda's algorithm is related to the size of the memory of the processes and we can expect to have smaller messages if we manage to adapt Mazurkiewicz's algorithm in the asynchronous message passing model.

Election and naming have also been studied in intermediate models between the models of Mazurkiewicz and of Yamashita and Kameda [Cha05, CM04, CMZ04, Maz04].

1.3. Our Results

We give a characterization of networks where naming and election can be solved in the asynchronous message passing model (Theorem 4.1). This result is based on the notions of fibrations and coverings.

Then, we present a characterization of graphs where election and naming can be solved for any port-numbering (Theorem 4.2).

In order to obtain necessary conditions, in Section 3 we introduce a way to encode any network with a port-numbering by a simple labelled digraph and we then obtain an impossibility result from the Angluin's lifting lemma [Ang80] (Proposition 3.2).

The naming algorithm $\mathcal{M}$ presented in Section 4 uses some ideas of the Mazurkiewicz's algorithm [Maz97]. It has some interesting properties that the previous existing algorithms do not have. Any execution of our algorithm needs a polynomial number of messages of polynomial size (Proposition 4.4), whereas the algorithms of Yamashita and Kameda and of Boldi *et al.* need messages of exponential size. Moreover, our algorithm is totally asynchronous, whereas the algorithms of Yamashita and Kameda and of Boldi *et al.* are executed in a pseudo-synchronous way. An interesting consequence of this property is that for any network $(\mathbf{G}, \nu)$, there exists an execution of our algorithm on $(\mathbf{G}, \nu)$ that enables to solve election and naming on $(\mathbf{G}, \nu)$ (Proposition 4.2). Thus, our algorithm may utilise the "asymmetry" of the execution even if the graph is really "symmetric".

2. Preliminaries

2.1. Undirected Graphs, Directed Graphs and Labelled (Di)Graphs

2.1.1. Undirected Graphs

We consider finite undirected connected graphs without multiple edges or loop called also simple graphs. Each such a graph is written as $G = (V(G), E(G))$ where $V(G)$ is the set of vertices of G and where the set of edges $E(G)$ is a set of pairs of distinct vertices of G. For each edge $\{u, v\} \in E(G)$, u and v are the *ends* of $\{u, v\}$ and u and v are said to be *adjacent* or *neighbours*. We denote by $N_G(u)$ the set of all vertices of G adjacent to u and $\deg_G(u)$ is the degree of u in G, i.e., the size of $N_G(u)$.

Throughout the paper we will consider graphs where vertices and edges are labelled with labels from a recursive set L. A graph G labelled over L will be denoted by (G, λ), where $\lambda\colon V(G) \cup E(G) \to L$ is the labelling function. The graph G is called the underlying graph and the mapping λ is a labelling of G. Labelled graphs will be designated by bold letters like $\mathbf{G}, \mathbf{H}, \ldots$ If $\mathbf{G}$ is a labelled graph, then G denotes the underlying graph.

We suppose that ϵ is a label that does not belong to L and then any partial labelling function λ of G defined on a set B of vertices and edges using labels from L can be canonically extended to a total labelling function of G by defining $\lambda(v) = \epsilon$ or $\lambda(e) = \epsilon$ for each vertex v or edge e in $(V(G) \cup E(G)) \setminus B$.

In some applications we need several labelling functions for a given graph G. Let $(\lambda_1, ..., \lambda_k)$ be a tuple of labelling functions of G, the labelled graph obtained with this tuple is denoted $(G, (\lambda_1, ..., \lambda_k))$ and the label of a vertex $v \in V(G)$ is $(\lambda_1(v), ..., \lambda_k(v))$.

2.1.2. Directed Graphs

In order to describe our characterization, one needs to consider also directed graphs (or digraphs) that can have multiple arcs and self-loops. A digraph D is defined by a set $V(D)$ of vertices, by a set $A(D)$ of arcs and by two maps s_D and t_D (in general, the subscripts will be omitted) from $A(D)$ to $V(D)$. For

each arc $a \in A(D)$, $s(a)$ is the *source* of a and $t(a)$ is its *target*. We say that a is going out of $s(a)$ and coming into $t(a)$. A self-loop is an arc with the same source and target. A digraph with no self-loop and such that for each couple (u, v) of vertices there is at most one arc a such that $s(a) = u$ and $t(a) = v$ is said to be simple.

A digraph D is *strongly connected* if for all vertices $u, v \in V(D)$, there exists a sequence of arcs $a_1, a_2, \dots a_p$ such that $s(a_1) = u, \forall i \in [1, p-1], t(a_i) = s(a_{i+1})$ and $t(a_p) = v$. In the following, we will only consider strongly connected digraphs.

A *symmetric* digraph D is a digraph endowed with a symmetry, that is, an involution $Sym : A \to A$ such that for every $a \in A : s(a) = t(Sym(a))$.

Definition 2.1. A *homomorphism* φ from the digraph D to the digraph D' is given by a pair of functions $\varphi_V \colon V(D) \to V(D')$ and $\varphi_A \colon A(D) \to A(D')$ commuting with the source and target maps, i.e., $s_{D'} \circ \varphi_A = \varphi_V \circ s_D$ and $t_{D'} \circ \varphi_A = \varphi_V \circ t_D$.

A homomorphism φ is an *isomorphism* if φ is bijective. We write $D \simeq D'$ whenever D and D' are isomorphic.

Throughout the paper we will consider digraphs where the vertices and the arcs are labelled with labels from a recursive set L. A digraph D labelled over L will be denoted by (D, λ), where $\lambda \colon V(D) \cup A(D) \to L$ is the labelling function. The digraph D is called the underlying digraph and the mapping λ is a labelling of D. A mapping $\varphi \colon V(D) \cup A(D) \to V(D') \cup A(D')$ is a homomorphism from (D, λ) to (D', λ') if φ is a digraph homomorphism from D to D' which preserves the labelling, i.e., such that $\lambda'(\varphi(x)) = \lambda(x)$ for every $x \in V(D) \cup A(D)$. Labelled digraphs will be designated by bold letters like $\mathbf{D}, \mathbf{D}', \dots$ If $\mathbf{D}$ is a labelled digraph, then D denotes the underlying digraph. As for graphs, one can note that it is possible to extend any partial labelling function λ of a digraph D using labels from a set L in a total labelling function of D by using a special label $\epsilon \notin L$.

Given a symmetric digraph D, the labelling of the arcs of D may reflect the fact that D is symmetric.

Definition 2.2. Let D be a digraph endowed with the symmetry Sym. let $\lambda : C \to C$ be a labelling function of D. The labelling λ is *symmetric* if there exists an involution $\iota : C \to C$ such that for each arc $a \in A(D)$, $\lambda(Sym(a)) = \iota(\lambda(a))$.

Given a labelled connected simple graph $\mathbf{G} = (G, \lambda)$, one associates a labelled symmetric strongly connected digraph denoted by $Dir(\mathbf{G}) = (Dir(G), \lambda)$ and defined as follows. The set of vertices of $Dir(G)$ is the set of vertices of G, i.e., $V(Dir(G)) = V(G)$ and each vertex of G has the same label in $Dir(G)$ as in G. For each edge $\{u, v\} \in E(G)$, there exists two arcs $a_{(u,v)}, a_{(v,u)} \in A(Dir(G))$ such that $s(a_{(u,v)}) = t(a_{(v,u)}) = u$, $t(a_{(u,v)}) = s(a_{(v,u)}) = v$ and $Sym(a_{(u,v)}) = a_{(v,u)}$. Note that this digraph does not contain multiple arcs or self-loop and that its arcs are unlabelled.

2.2. Fibrations and coverings

The notions of fibrations and coverings are fundamental in this work. Definitions, main properties and some applications are presented in [Bod89, BV02].

A fibration is a homomorphism that induces an isomorphism between the incoming arcs of each vertex and the incoming arcs of its image.

Definition 2.3. A digraph D is *fibred over* a digraph D' via a homomorphism φ if φ is a homomorphism from D to D' such that for each arc $a' \in A(D')$ and for each vertex $v \in \varphi^{-1}(t(a'))$, there exists a unique arc $a \in A(D)$ such that $t(a) = v$ and $\varphi(a) = a'$; this arc a is called the *lifting* of a' at v.

We say that the homomorphism φ is a *fibration* from D to D', the digraph D is the *total digraph* of φ and the digraph D' is the *base* of φ.

The *fibre* over a vertex v' (resp. an arc a') of D' is defined as the set $\varphi^{-1}(v')$ of vertices of D (resp. the set $\varphi^{-1}(a')$ of arcs of D).

In the sequel digraphs are always strongly connected and total digraphs non empty.

A covering projection is a fibration that also induces an isomorphism between the outgoing arcs of each vertex and the outgoing arcs of its image.

Definition 2.4. A digraph D is a *covering* of a digraph D' via a homomorphism φ if φ is a homomorphism from D to D' such that for each arc $a' \in A(D')$ and for each vertex $v \in \varphi^{-1}(t(a'))$ (resp. $v \in \varphi^{-1}(s(a'))$, there exists a unique arc $a \in A(D)$ such that $t(a) = v$ (resp. $s(a) = v$) and $\varphi(a) = a'$.

The homomorphism φ is called a *covering projection* from D to D'.

In the sequel a covering projection will be called a covering. If D' has no self-loop and no multiple arcs then the covering is said to be simple.

A symmetric covering is a covering between symmetric digraphs that preserves the function Sym.

Definition 2.5. A symmetric digraph D is called a *symmetric covering* of a symmetric digraph D' via a homomorphism φ if D is a covering of D' via φ and if for each arc $a \in A(D)$, $\varphi(Sym(a)) = Sym(\varphi(a))$.

The homomorphism φ is called a *symmetric covering* from D to D'.

A symmetric digraph D is said to be *symmetric covering prime* if for each symmetric digraph D' such that D is a symmetric covering of D', $D \simeq D'$.

All these definitions are extended in a natural way to labelled digraphs.

An interesting property satisfied by covering is that all the fibres have the same cardinality, that is called the *number of sheets* of the covering. The following proposition is a result of Boldi and Vigna [BV02].

Proposition 2.1. A covering $\varphi : \mathbf{D} \to \mathbf{D}'$ with a connected base and a nonempty covering is surjective; moreover, there exists $q \in \{1, 2, \cdots\}$ such that for each $x \in V(D') \cup A(D')$, $|\varphi^{-1}(x)| = q$.

Given a labelled digraph $\mathbf{D}$, there exists a "minimal" labelled digraph $\mathbf{D}_0$ such that $\mathbf{D}$ is fibred over $\mathbf{D}_0$. The existence of such a labelled digraph has been shown by Boldi and Vigna [BV02].

Proposition 2.2. For any strongly connected labelled digraph $\mathbf{D}$, there exists a strongly connected labelled digraph $\mathbf{D}_0$ such that $\mathbf{D}$ is fibred over $\mathbf{D}_0$ and such that for any strongly connected labelled digraph $\mathbf{D}'$, if $\mathbf{D}$ is fibred over $\mathbf{D}'$, then $\mathbf{D}'$ is fibred over $\mathbf{D}_0$.

The digraph $\mathbf{D}_0$ is called the minimum base of $\mathbf{D}$.

The minimum base of a labelled digraph $\mathbf{D}$ can be computed in polynomial time using the degree refinement technique [BV02, Lei82]; this method is close to the technique used to minimize a deterministic automaton [HU79].

Let D and D' be two digraphs such that D is a surjective covering of D' via φ. If D' has no self-loop then for each arc $a \in A(D) : \varphi(s(a)) \neq \varphi(t(a))$. Finally, the following property is a direct consequence of the definitions and it is fundamental in the sequel of this paper :

Proposition 2.3. Let $\mathbf{D}$ and $\mathbf{D'}$ be two labelled digraphs such that $\mathbf{D'}$ has no self-loop and $\mathbf{D}$ is a surjective covering of $\mathbf{D'}$ via φ. Let a_1 and a_2 be two arcs of $\mathbf{D}$. If $a_1 \neq a_2$ and $a_1, a_2 \in \varphi^{-1}(a')$ $(a' \in A(D'))$ then $\{s(a_1), t(a_1)\} \cap \{s(a_2), t(a_2)\} = \emptyset$.

2.3. Local Computations on Arcs

In this paper we consider labelled digraphs and we assume that local computations modify only labels of vertices and of arcs. Digraph relabelling systems on arcs and more generally local computations on arcs satisfy the following constraints, that arise naturally when describing distributed computations with decentralized control:

$(C1)$ they do not change the underlying digraph but only the labelling of vertices and of the arcs, the final labelling being the result of the computation (*relabelling relations*),

$(C2)$ they are *local*, that is, each relabelling step changes only the label of the source, the label of the target of an arc and the label of the arc,

$(C3)$ they are *locally generated*, that is, the applicability of a relabelling rule on an arc only depends on the label of the arc, the labels of the source and of the target (locally generated relabelling relation).

The relabelling is performed until no more transformation is possible, i.e., until a normal form is obtained.

The more formal framework is the following. Let $\mathcal{D}_L$ be the class of L-labelled digraphs. Then any binary relation $\mathcal{R} \subseteq \mathcal{D}_L \times \mathcal{D}_L$ on $\mathcal{D}_L$ is called a *digraph rewriting relation*. We assume that it is closed under isomorphism, i.e., if $\mathbf{D}\ \mathcal{R}\ \mathbf{D}_1$ and $\mathbf{D'} \simeq \mathbf{D}$ then $\mathbf{D'}\ \mathcal{R}\ \mathbf{D'_1}$ for some labelled digraph $\mathbf{D'_1} \simeq \mathbf{D_1}$. In the remainder of the paper $\mathcal{R}^*$ stands for the reflexive-transitive closure of $\mathcal{R}$. The labelled digraph $\mathbf{D}$ is *$\mathcal{R}$-irreducible* (or just irreducible if $\mathcal{R}$ is fixed) if there is no $\mathbf{D_1}$ such that $\mathbf{D}\ \mathcal{R}\ \mathbf{D_1}$.

Definition 2.6. Let $\mathcal{R} \subseteq \mathcal{D}_L \times \mathcal{D}_L$ be a digraph rewriting relation. The relation $\mathcal{R}$ is a *relabelling relation* if whenever two labelled digraphs are in relation then the underlying unlabelled digraphs are equal, i.e.,$\mathbf{D}_1\ \mathcal{R}\ \mathbf{D}_2$ implies that $D_1 = D_2$.

Definition 2.7. Let $\mathcal{R} \subseteq \mathcal{D}_L \times \mathcal{D}_L$ be a digraph relabelling relation. The relation $\mathcal{R}$ is (arc) *local* if $(D, \lambda)\ \mathcal{R}\ (D, \lambda')$ implies that there exists an arc $a \in A(D)$ such that $\lambda(x) = \lambda'(x)$ for every $x \notin \{a, s(a), t(a)\}$.

The next definition states that an arc local relabelling relation $\mathcal{R}$ is *arc locally generated* if the applicability of any relabelling depends only on the labels of the arc and of the ends of the arc.

Definition 2.8. Let $\mathcal{R}$ be a relabelling relation arc local. Then $\mathcal{R}$ is *arc locally generated* if the following is satisfied: for all labelled digraphs (D_1, λ), (D_1, λ'), (D_2, η), (D_2, η') and all arcs $(u, u') \in A(D_1)$ and $(v, v') \in A(D_2)$, the following three conditions:

1. $\lambda(u) = \eta(v)$, $\lambda(u') = \eta(v')$, $\lambda'(u) = \eta'(v)$, $\lambda'(u') = \eta'(v')$,
2. $\lambda(w) = \lambda'(w)$, for each $w \in V(D_1) \setminus \{u, u'\}$,
3. $\eta(w) = \eta'(w)$, for each $w \in V(D_2) \setminus \{v, v'\}$,

imply that $(D_1, \lambda)\ \mathcal{R}\ (D_1, \lambda')$ if and only if $(D_2, \eta)\ \mathcal{R}\ (D_2, \eta')$.

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, *arc local computations* are computations on labelled digraphs corresponding to arc locally generated relabelling relations.

Given an arc locally generated relation $\mathcal{R}$, a *computation step* on a digraph $\mathbf{D}$ is the relabelling of the labels of an arc a and of the ends of a that leads to a digraph $\mathbf{D}'$ such that $\mathbf{D}\ \mathcal{R}\ \mathbf{D}'$; the arc a is the support of the relabelling step. An *execution* of $\mathcal{R}$ on $\mathbf{D}$ is a sequence $\mathbf{D} = \mathbf{D}_0\ \mathcal{R}\ \mathbf{D}_1\ \mathcal{R}\ \ldots\ \mathcal{R}\ \mathbf{D}_i\ \mathcal{R}\ \ldots$ where $\mathbf{D}_i$ is called the *configuration* of D at step i. A *final* configuration is a configuration $\mathbf{D}$ where no more relabelling step can be applied, i.e., there does not exists any $\mathbf{D}'$ such that $\mathbf{D}\ \mathcal{R}\ \mathbf{D}'$.

A complete presentation of graph relabelling systems and local computations for the model of Mazurkiewicz may be found in [GMM04] (pp. 256-260). It can be easily adapted to the model studied in this work.

2.4. Distributed Computations of Local Computations on Arcs

The notion of relabelling sequence defined above obviously corresponds to a notion of *sequential* computation. Clearly, a locally generated relabelling relation allows parallel relabellings too, since non-overlapping edges may be relabelled independently. Thus we can define a distributed way of computing by allowing that two consecutive relabelling steps with disjoint supports may be applied in any order (or concurrently). More generally, any two relabelling sequences such that one can be obtained from the other by exchanging successive concurrent steps, lead to the same result.

Hence, the notion of relabelling sequence associated to a arc locally generated relabelling relation may be regarded as a *serialization* of a distributed computation. This model is asynchronous, in the sense that several relabelling steps *may* be done at the same time but we do not require that all of them have to be performed. In the sequel we will essentially handle sequential relabelling sequences, but the reader should keep in mind that such sequences may be done in parallel.

2.5. Coverings and Arc Local Computations

First, We present a fundamental lemma which connects coverings and locally generated relabelling relations on arcs. It is the natural extension of the lifting lemma [Ang80]. It states that whenever $\mathbf{D}_1$ is a covering of $\mathbf{D}'_1$, every relabelling step in $\mathbf{D}'_1$ can be lifted to a relabelling chain in $\mathbf{D}_1$ which is compatible with the covering relation. It is a direct consequence of Proposition 2.3.

Lemma 2.1. (lifting lemma)
Let $\mathcal{R}$ be a locally generated relabelling relation on arcs and let $\mathbf{D}_1$ be a covering of the labelled digraph $\mathbf{D}'_1$ via the morphism γ; we assume that $\mathbf{D}'_1$ has no self-loop. If $\mathbf{D}'_1\ \mathcal{R}^*\ \mathbf{D}'_2$ then there exists $\mathbf{D}_2$ such that $\mathbf{D}_1\ \mathcal{R}^*\ \mathbf{D}_2$ and $\mathbf{D}_2$ is a covering of $\mathbf{D}'_2$ via γ.

Proof:
Consider two labelled digraphs $\mathbf{D}_1 = (D_1, \lambda)$ and $\mathbf{D}'_1 = (D'_1, \eta)$ such that $\mathbf{D}_1$ is a covering of $\mathbf{D}'_1$ via a homomorphism γ and an arc locally generated relabelling relation $\mathcal{R}$.

It is sufficient to prove the lemma when $\mathbf{D}'_1 = (D'_1, \eta)\ \mathcal{R}\ (D'_1, \eta') = \mathbf{D}'_2$, let r be the rewritting rule for this step. Consider the rewritten arc $a = (v, v') \in A(D'_1)$: for each $x \in (V(D'_1) \cup A(D'_1)) \setminus \{a, v, v'\}$, $\eta'(x) = \eta(x)$. Since $\mathbf{D}_1$ is a covering of $\mathbf{D}'_1$ via γ and $\mathbf{D}'_1$ has no self-loop from Proposition 2.3 arcs of $A(D_1)$ which are mapped by γ on (v, v') are disjoint. Consequently, one can apply the relabelling rule r on each arc $(u, u') \in A(D_1)$ such that $\gamma(u) = v$ and $\gamma(u') = v'$. Let λ' be the new labelling of D_1 obtained once all these relabelling steps have been performed. For each $u \in \gamma^{-1}(\{v, v'\})$, $\lambda'(u) = \eta'(\gamma(u))$ and for each $u \in V(D_1) \setminus \gamma^{-1}(\{v, v'\})$, $\lambda'(u) = \lambda(u) = \eta(\gamma(u)) = \eta'(\gamma(u))$. The same relations hold for arcs. Thus, the labelled digraph $\mathbf{D}_2 = (D_1, \lambda')$ is a covering of $\mathbf{D}'_2$ via γ. □

3. From Asynchronous Message Passing to Local Computations on Arcs

3.1. The Asynchronous Message Passing Model

Our model follows standard models for distributed systems given in [AW04, Lyn96, Tel00]. The communication model is a point-to-point communication network which is represented as a simple connected undirected graph where vertices represent processes and two vertices are linked by an edge if the corresponding processes have a direct communication link. Processes communicate by message passing and each process can distinguish its neighbours, i.e., the different links incident to it.

Since each process knows from which channel it receives a message or to which channel it sends a message, one supposes that the network is represented by a simple graph with a port-numbering function.

Definition 3.1. Given a simple labelled graph $\mathbf{G}$, a *port-numbering* function ν is a set of local functions $\{\nu_u \mid u \in V(G)\}$ such that for each vertex $u \in V(G)$, ν_u is a bijection between $N_G(u)$ and $[1, \deg_G(u)]$.

Remark 3.1. We consider graphs with a port-numbering thus information concerning edges may be attached to vertices and we don't need labels on edges.

We assume in the sequel that edges of $\mathbf{G}$ have no labels. We consider the asynchronous message passing model: processes cannot access a global clock, processes execute computation steps at arbitrary speed, and a message sent from a process to a neighbour arrives within some finite but unpredictable time.

3.2. Port-numbering and Symmetric Digraphs

A network is represented by a simple labelled graph $(\mathbf{G}, \nu)$ where $\mathbf{G} = (G, \lambda)$ is a simple graph whose vertices are labelled and ν is a port-numbering function.

Remark 3.2. The labelling λ of processes may encode some properties of the network. For example, if the network is anonymous, all the vertices have the same label (i.e., $\forall u, u' \in V(G), \lambda(u) = \lambda(u')$). If the processes have unique identities, then for all $u, u' \in V(G)$ if $u \neq u'$ then $\lambda(u) \neq \lambda(u')$. If there exists a distinguished process, then there exists $u \in V(G)$ such that for each $u' \in V(G)$ distinct from u, $\lambda(u) \neq \lambda(u')$.

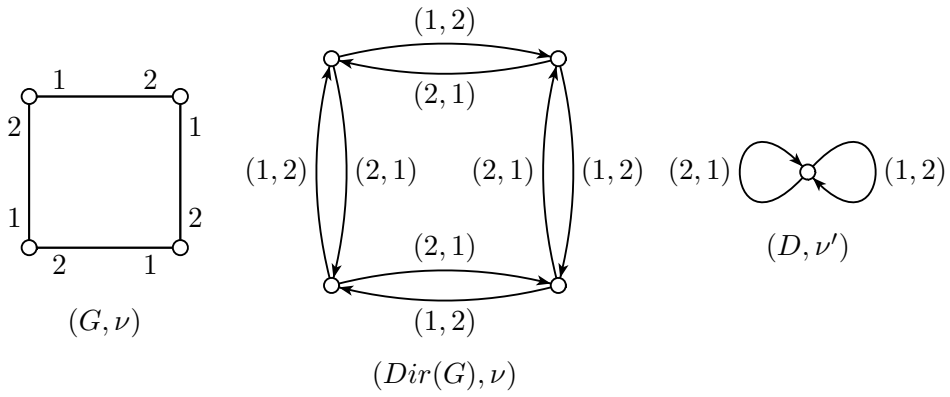

Figure 1. A graph G with a port numbering ν and the associated labelled digraph $(Dir(G), \nu)$ that is a covering of (D, ν'). From Proposition 3.2, there is no election algorithm for (G, ν). The same argument gives the same result for any ring.

Given a network $(\mathbf{G}, \nu)$, one associates to $(\mathbf{G}, \nu)$ a symmetric labelled digraph $(Dir(\mathbf{G}), \nu)$ where each vertex $v \in V(Dir(G))$ has the same label as in $\mathbf{G}$ and where each arc $a_{(u,v)} \in A(Dir(G))$ such that $s(a) = u$ and $t(a) = v$ is labelled by $(\nu_u(v), \nu_v(u))$. Let ι be defined by : $\iota((p, q)) = (q, p)$, where p and q are two integers. One can note that for each arc $a \in A(Dir(G))$ labelled by (p, q), the arc $Sym(a)$ is labelled by $(q, p) = \iota((p, q))$, i.e., the labelling of the arcs of $Dir(G)$ induced by ν is symmetric.

Examples of this construction are presented on Figures 1, 2 and 3.

Remark 3.3. Let (G, ν) be a simple graph endowed with a port-numbering ν. The labelled digraph $(Dir(G), \nu)$ is symmetric covering prime if and only if $(Dir(G), \nu)$ considered as an automaton whose all states are final is minimal.

3.3. Encoding a Network with a Labelled Digraph

The construction presented in this section may appear technical nevertheless the intuition is very natural and simple, and it is illustrated in Figures 2 and 3.

Given a labelled digraph $\mathbf{D} = (D, \lambda)$ whose arcs are unlabelled, one will associate to $\mathbf{D}$ a simple labelled digraph $\overleftrightarrow{\mathbf{D}}$ defined in the following way.

To each arc $a \in A(D)$ whose source is u and whose target is v, we associate the set V_a of three vertices denoted $\{outbuf_a(u, v), canal_a(u, v), inbuf_a(u, v)\}$ and the set A_a of four arcs that are: $(u, outbuf_a(u, v)), (outbuf_a(u, v), canal_a(u, v)), (canal_a(u, v), inbuf_a(u, v)), (inbuf_a(u, v), v)$.

The unlabelled simple digraph $\overleftrightarrow{D}$ is then defined by:

$$V(\overleftrightarrow{D}) = V(D) \cup (\bigcup_{a \in A(D)} V_a) \text{ and } A(\overleftrightarrow{D}) = \bigcup_{a \in A(D)} A_a.$$

We need to memorize the meaning (semantic) of the different vertices of $\overleftrightarrow{D}$. To do so, we consider the labelled digraph $\overleftrightarrow{\mathbf{D}} = (\overleftrightarrow{D}, (\kappa, \lambda))$ where λ and κ are two labellings of the vertices of $\overleftrightarrow{D}$ defined as

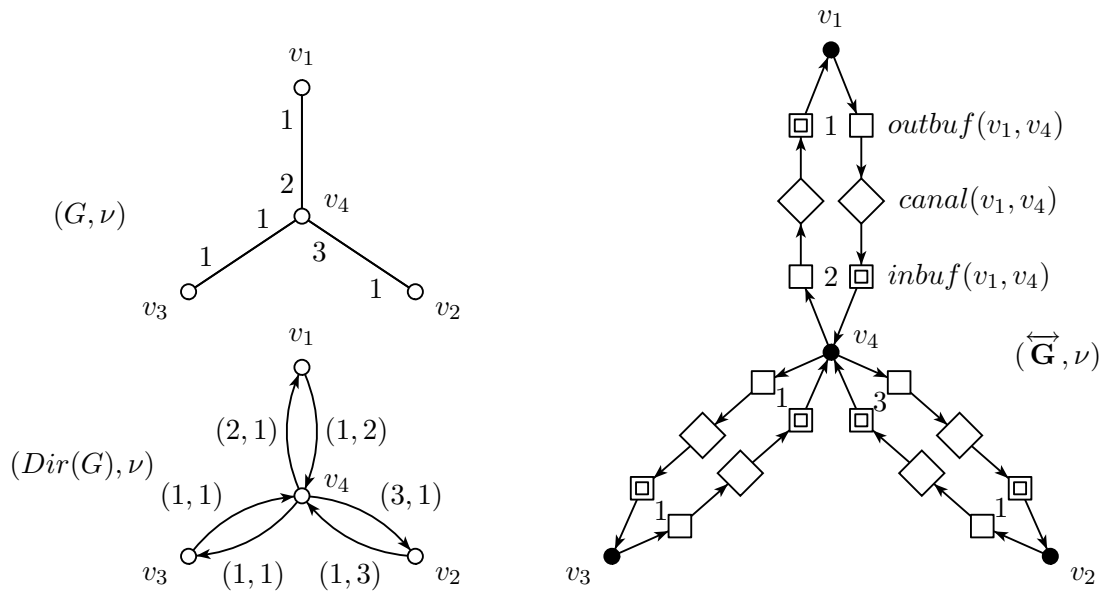

Figure 2. We adopt the following notation for vertices of $(\overleftrightarrow{\mathbf{G}}, \nu)$. A black-circle vertex corresponds to the label **process**, a square vertex corresponds to the label **send**, a diamond vertex corresponds to the label **transmission**, and a double-square vertex corresponds to the label **receive** There is no multiple arcs thus subscripts for $outbuf$, $canal$ and $inbuf$ are omitted.

follows. For each vertex $v \in V(\overleftrightarrow{D})$, $\lambda(v)$ is the label of v in $\mathbf{D}$ if $v \in V(D)$ and $\lambda(v) = \epsilon$ otherwise. The labelling function κ encodes the role of each vertex of $\overleftrightarrow{D}$ and is defined as follows:

- $\forall u \in V(D)$, $\kappa(u) = \mathbf{process}$,
- $\forall a \in A(D)$, $\kappa(outbuf_a(s(a), t(a))) = \mathbf{send}$,
- $\forall a \in A(D)$, $\kappa(inbuf_a(s(a), t(a))) = \mathbf{receive}$,
- $\forall a \in A(D)$, $\kappa(canal_a(s(a), t(a))) = \mathbf{transmission}$.

One considers now a labelling function ν of the arcs of $\mathbf{D}$ such that for each arc $a \in V(D)$, there exists positive integers p, q such that $\nu(a) = (p, q)$. We extend ν into a labelling function of vertices of $\overleftrightarrow{D}$ such that for each arc $a \in A(D)$ whose label is (p, q), $\nu(outbuf_a(a)) = p$ and $\nu(inbuf_a(a)) = q$ (for the other vertices, ν is equal to ϵ).

Suppose now that $\mathbf{D}$ is a symmetric labelled digraph and that ν is a symmetric labelling function of $\mathbf{D}$. One notices that for each arc $a \in A(D)$, $\nu(inbuf_a(a)) = \nu(outbuf_{Sym(a)}(Sym(a)))$.

Given a network $(\mathbf{G}, \nu)$ where $\mathbf{G}$ is a simple labelled graph and ν is a port-numbering of $\mathbf{G}$, we note $(\overleftrightarrow{\mathbf{G}}, \nu)$ the simple labelled digraph obtained by applying the construction described above on the labelled digraph $(Dir(\mathbf{G}), \nu)$.

3.4. Encoding Basic Instructions with Local Computations on Arcs

As in [YK96] (see also [Tel00] pp. 45-46), we assume that each process, depending on its state, either changes its state, or receives a message via a port or sends a message via a port.

Given a network $(\mathbf{G}, \nu)$ and its representation $(\overleftrightarrow{\mathbf{G}}, \nu)$ as a simple labelled digraph described in the previous section, we now explain how the basic instructions each process of $(\mathbf{G}, \nu)$ can execute in terms of local computations on arcs of $(\overleftrightarrow{\mathbf{G}}, \nu)$:

- an event that enables a process to modify its state (i.e., an internal transition) is encoded by a relabelling rule that can be applied on a vertex $v \in V(\overleftrightarrow{G})$ such that $\kappa(v) = \mathbf{process}$,
- a send event of the form "send a message $\mathbf{m}$ via port p" is encoded by a relabelling rule that can be applied on the arc $(u, v) \in A(\overleftrightarrow{G})$ where $\kappa(u) = \mathbf{process}$, $\kappa(v) = \mathbf{send}$ and $\nu(v) = p$,
- a receive event of the form "receive a message $\mathbf{m}$ via port q" is encoded by a relabelling rule that can be applied on the arc $(v, u) \in A(\overleftrightarrow{G})$ where $\kappa(u) = \mathbf{process}$, $\kappa(v) = \mathbf{receive}$ and $\nu(v) = q$,
- an event concerning the transmission control can be encoded by a relabelling rule concerning an arc of the form (u, v) or (v, u) with $\kappa(u) \in \{\mathbf{send}, \mathbf{receive}\}$ and $\kappa(v) = \mathbf{transmission}$.

Proposition 3.1. Let $\mathbf{D}$ and $\mathbf{D}'$ be two symmetric labelled digraphs. Let ν (resp. ν') be a symmetric labelling function of the arcs of $\mathbf{D}$ (resp. $\mathbf{D}'$) such that for each arc $a \in V(D)$, there exists positive integers p, q such that $\nu(a) = (p, q)$ and for each arc $a' \in V(D')$, there exists positive integers p, q such that $\nu'(a') = (p, q)$. If $(\mathbf{D}, \nu)$ is a covering of $(\mathbf{D}', \nu')$ then $(\overleftrightarrow{\mathbf{D}}, \nu)$ is a covering of $(\overleftrightarrow{\mathbf{D}}', \nu')$.

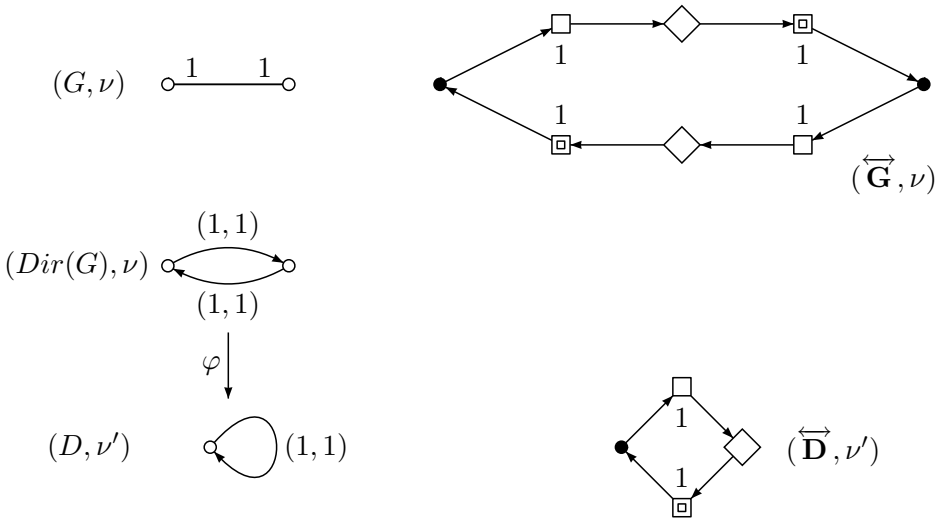

Figure 3. The digraph $(Dir(G),\nu)$ is a proper symmetric covering of (D,ν') and thus there is no election algorithm for (G,ν) from Proposition 3.2.

Proof:
Consider two digraphs $((D,\lambda),\nu),((D',\lambda'),\nu')$ such that $(\mathbf{D},\nu)$ is a covering of $(\mathbf{D}',\nu')$ via a homomorphism φ. We define a homomorphism γ from $(\overleftrightarrow{\mathbf{D}},\nu)$ to $(\overleftrightarrow{\mathbf{D}}',\nu')$ as follows. For each vertex $v \in V(\overleftrightarrow{D})$, we define $\gamma(v)=\varphi(v)$ if $v \in V(D)$. For each arc $a \in A(D)$ such that $s(a)=u$ and $t(a)=v$, we define $\gamma(outbuf_a(u,v)) = outbuf_{\gamma(a)}(\gamma(u),\gamma(v))$, $\gamma(canal_a(u,v)) = canal_{\gamma(a)}(\gamma(u),\gamma(v))$ and $\gamma(inbuf_a(u,v)) = inbuf_{\gamma(a)}(\gamma(u),\gamma(v))$.

It is then easy to check that $(\overleftrightarrow{\mathbf{D}},\nu)$ is a simple covering of $(\overleftrightarrow{\mathbf{D}}',\nu')$ via the the homomorphism γ. □

3.5. A Necessary Condition for the Election

We present here a necessary condition that must be satisfied by any network $(\mathbf{G},\nu)$ that admits an election algorithm.

Proposition 3.2. Given a simple labelled graph $\mathbf{G}$ and a port-numbering function ν of $\mathbf{G}$. If there exists an election or a naming algorithm for $\mathbf{G}$ endowed with ν as port-numbering then $(Dir(\mathbf{G}),\nu)$ is symmetric covering prime.

Proof:
By contradiction. Consider a simple labelled graph $\mathbf{G}$, a port-numbering function ν of $\mathbf{G}$ and a labelled digraph $(\mathbf{D},\nu')$ such that $(Dir(\mathbf{G}),\nu)$ is a proper symmetric covering of $(\mathbf{D},\nu')$ via a homomorphism γ. From Proposition 3.1, we know that $(\overleftrightarrow{\mathbf{G}},\nu)$ is a covering of $(\overleftrightarrow{\mathbf{D}},\nu')$.

Consider now a message passing algorithm $\mathcal{A}$ on $(\mathbf{G},\nu)$ and the corresponding algorithm $\mathcal{A}'$ on $(\overleftrightarrow{\mathbf{G}},\nu)$ using local computations on arcs obtained by the transformation described in Section 3.4.

Note that if there exists a no finite execution of $\mathcal{A}'$ on $(\overleftrightarrow{\mathbf{D}}, \nu')$, then there exists a no finite execution of $\mathcal{A}'$ on $(\overleftrightarrow{\mathbf{G}}, \nu)$, and then there exists a no finite execution of $\mathcal{A}$ on $(\mathbf{G}, \nu)$. Finally, $\mathcal{A}$ is not an election (or naming) algorithm for $(\mathbf{G}, \nu)$.

Consider a finite execution ρ of $\mathcal{A}'$ on $(\overleftrightarrow{\mathbf{D}}, \nu')$. The labelled digraph $(\overleftrightarrow{\mathbf{D}}, \nu')$ has no self-loop and, from Lemma 2.1, there exists an execution ρ' of $\mathcal{A}'$ on $(\overleftrightarrow{\mathbf{G}}, \nu)$ that is lifted up from this finite execution of $\mathcal{A}'$ on $(\overleftrightarrow{\mathbf{D}}, \nu')$. Consequently, as the covering is strict, from Proposition 2.1, each label that appears in the final configuration of ρ in $(\overleftrightarrow{\mathbf{D}}, \nu')$ appears at least twice in the final configuration of ρ' in $(\overleftrightarrow{\mathbf{G}}, \nu)$.

Consequently, there exists an execution of $\mathcal{A}$ on $(\mathbf{G}, \nu)$, where in the final configuration, no vertex has a unique label. Consequently, $\mathcal{A}$ is not an election (or naming) algorithm for $(\mathbf{G}, \nu)$. □

Remark 3.4. As immediate consequences of this result we deduce two classical results for the asynchronous message passing model: there exists no election algorithm in an anonymous network of two processes ([Tel00] p. 316), and there exists no algorithm for election in an anonymous ring of known size ([Tel00] Theorem 9.5 p. 317) (sketches of the proofs are given in Figures 1 and 3).

4. A Mazurkiewicz-like Algorithm

We show in this section that the necessary condition given by the Proposition 3.2 is also sufficient. To do so, we present an enumeration algorithm inspired by Mazurkiewicz's algorithm and adapted to the message passing model.

We first give a general description of our algorithm, that will be denoted $\mathcal{M}$, when executed on a connected labelled simple graph $\mathbf{G}$ with a port-numbering ν.

During the execution of the algorithm, each vertex v attempts to get its own unique identity which is a number between 1 and $|V(G)|$. Once a vertex u has chosen a number $n(v)$, it sends it to each neighbour u with the port-number $\nu_u(v)$. When a vertex v receives a message from one neighbour v, it stores the number $n(v)$ with the port numbers $\nu_u(v)$ and $\nu_v(u)$. From all information it has gathered from its neighbours, each vertex can construct its *local view* (which is the set of numbers of its neighbours associated with the corresponding port numbers). Then, a vertex broadcasts its number with its *local view*. 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 symmetric covering prime, then every vertex will have a unique number: the algorithm is a naming algorithm.

4.1. Labels

We consider a network $(\mathbf{G}, \nu)$ where $\mathbf{G} = (G, \lambda)$ is a simple labelled graph and where ν is a port-numbering of $\mathbf{G}$. The function $\lambda : V(G) \to L$ is the initial labelling and is not modified during the execution of the algorithm. We suppose that there exists a total order $<_L$ on L. During the execution, the label of each vertex v is a tuple $(\lambda(v), n(v), N(v), M(v))$ representing following information.

- $\lambda(v) \in L$ is the initial label of v and is not modified by the algorithm.
- $n(v) \in \mathbb{N}$ is the current *number* of the vertex v computed by the algorithm.

- $N(v) \in \mathcal{P}_{\text{fin}}(\mathbb{N}^3)$[1] is the *local view* of v. The local view of a vertex v contains the information a vertex v has about its neighbours. If a vertex v has a neighbour u such that $\nu_u(v) = p$ and $\nu_v(u) = q$, then $(m, p, q) \in N(v)$ if the last message that v gets from u indicates that $n(u) = m$.

- $M(v) \in \mathbb{N} \times L \times \mathcal{P}_{\text{fin}}(\mathbb{N}^3)$ is the *mailbox* of v. The mailbox of v contains all information received by v during the execution of the algorithm. If $(m, \ell, \mathcal{N}) \in M(v)$, it means that at some previous step of the execution, there was a vertex u such that $n(u) = m$, $\lambda(u) = \ell$ and $N(u) = \mathcal{N}$.

Initially, each vertex v has a label of the form $(\lambda(v), 0, \emptyset, \emptyset)$ indicating that it has not choosen any number, that it has no information about its neighbours or about the other vertices of the graph.

In our algorithm, processes exchange messages of the form $< (m, n_{old}, M), p >$. If a vertex u sends a message $< (m, n_{old}, M), p >$ to one of its neighbour v, then the message contains following information.

- m is the current number $n(u)$ of u.

- n_{old} is the previous number of u, i.e., the number u sends to v in its previous message; if in the meanwhile, u has not modified its number, then $n_{old} = m$.

- M is the mailbox of u.

- p is the port-number the message has been sent through, i.e., $p = \nu_u(v)$.

4.2. An Order on Local Views

As in Mazurkiewicz's algorithm [Maz97], the nice properties of the algorithm rely on a total order on local views, i.e., on finite subsets of $\mathbb{N}^3$. We consider the usual lexicographic order on $\mathbb{N}^3$: $(n, p, q) < (n', p', q')$ if $n < n'$, or if $n = n'$ and $p < p'$, or if $n = n'$, $p = p'$ and $q < q'$.

Then, we use the same order on finite sets as Mazurkiewicz: given two distinct sets $N_1, N_2 \in \mathcal{P}_{\text{fin}}(\mathbb{N}^3)$, we define $N_1 \prec N_2$ if the maximum of the symmetric difference $N_1 \triangle N_2 = (N_1 \setminus N_2) \cup (N_2 \setminus N_1)$ belongs to N_2.

If $N(u) \prec N(v)$, then we say that the local view $N(v)$ of v is *stronger* than the local view $N(u)$ of u and that $N(u)$ is *weaker* than $N(v)$. Using the total order $<_L$ on L, we define $(\ell, \mathcal{N}) \prec (\ell', \mathcal{N}')$ if either $\ell <_L \ell'$, or $\ell = \ell'$ and $\mathcal{N} \prec \mathcal{N}'$. We denote by $\preceq$ the reflexive closure of $\prec$.

4.3. The Enumeration Algorithm $\mathcal{M}$

The algorithm for the vertex v_0 (see Algorithm 1) is expressed in an event-driven description (see Tel [Tel00] p. 553). The algorithm we describe here requires that the messages sent in a communication channel arrive in the same order they are sent (FIFO property of communication channels); we will explain in Section 4.5 how to drop this hypothesis.

The action **I** can be executed by a process on wake-up only if it has not received any message. In this case, it chooses the number 1, updates its mailbox and informs its neighbours.

The action **R** describes the instructions the vertex v_0 has to follow when it receives a message $< (n', n'_{old}, M'), p >$ from a neighbour via port q. First, it updates its mailbox by adding M' to it. Then it

[1] For any set S, $\mathcal{P}_{\text{fin}}(S)$ denotes the set of finite subsets of S.

modifies its number if there exists $(n(v_0), \ell, \mathcal{N}) \in M(v_0)$ such that $(\lambda(v_0), N(v_0)) \prec (\ell, \mathcal{N})$. Then, it updates its local view by removing (n'_{old}, p, q) from $N(v_0)$ (if $N(v_0)$ contains such an element) and by adding (n', p, q) to $N(v_0)$. Then, it adds its new state $(n(v_0), \lambda(v_0), N(v_0))$ to its mailbox. Finally, if its mailbox has been modified by the execution of all these instructions, it sends its number and its mailbox to all its neighbours.

If the mailbox of v_0 is not modified by the execution of the action $\mathbf{R}$, it means that the information v_0 has about its neighbour (i.e., its number) was correct, that all the elements of M' already belong to $M(v_0)$, and that for each $(n(v_0), \ell, \mathcal{N}) \in M(v_0)$, $(\ell, \mathcal{N}) \preceq (\lambda(v_0), N(v_0))$.

Algorithm 1: Algorithm $\mathcal{M}$.

```
I : {n(v_0) = 0 and no message has arrived at v_0}
begin
    n(v_0) := 1 ;
    M(v_0) := {(n(v_0), λ(v_0), ∅)};
    for i := 1 to deg(v_0) do
        send < (n(v_0), 0, M(v_0)), i > via port i ;
end

R : {A message < (n', n'_old, M'), p > has arrived at v_0 from port q}
begin
    M_old := M(v_0);
    n_old := n(v_0);
    M(v_0) := M(v_0) ∪ M';
    if n(v_0) = 0 or ∃(n(v_0), ℓ, N) ∈ M(v_0) such that (λ(v_0), N(v_0)) ≺ (ℓ, N) then
        n(v_0) := 1 + max{n | ∃(n, ℓ, N) ∈ M(v_0)};
    N(v_0) := N(v_0) \ {(n'_old, p, q)} ∪ {(n', p, q)};
    M(v_0) := M(v_0) ∪ {(n(v_0), λ(v_0), N(v_0))};
    if M(v_0) ≠ M_old then
        for i := 1 to deg(v_0) do
            send < (n(v_0), n_old, M(v_0)), i > through port i;
end
```

4.4. Correctness of $\mathcal{M}$

We consider a simple connected labelled graph $\mathbf{G}$ and a port-numbering ν of $\mathbf{G}$. For each vertex v, an internal transition is the execution of the instructions that update its state once a message has been received, but not the send events that follow this transition. If a vertex v executes the action $\mathbf{I}$, the corresponding internal transition is the modification of its number and of its mailbox.

An execution of the algorithm on $(\mathbf{G}, \nu)$ is a sequence of send events, receive events and internal events, where at each step, one transition is performed (this corresponds to the notion of execution defined by Tel [Tel00] (pp. 45-47) and thus an execution is seen as a serialization of the distributed execution).

We consider an execution ρ of $\mathcal{M}$ on $(\mathbf{G}, \nu)$ and for each vertex $v \in V(G)$, we denote by $(\lambda(v), n_i(v), N_i(v), M_i(v))$ the state of v after the ith computation step of ρ. We first remark that the steps that correspond to send events and receive events do not modify the value of $(\lambda(v), n(v), N(v), M(v))$ for any vertex $v \in V(G)$. Moreover, for each computation step where an internal event is performed, the value of $(\lambda(v), n(v), N(v), M(v))$ is modified for at most one vertex $v \in V(G)$.

The following lemma can be easily proved by an induction on the length of the execution and summarizes some simple properties that each execution of the algorithm $\mathcal{M}$ satisfies.

Lemma 4.1. For each vertex $v \in V(G)$ and for each step i,

1. $\exists (n,p,q) \in N_i(v) \iff \exists v' \in N_G(v)$ such that $\nu_v(v') = q$ and $\nu_{v'}(v) = p$,
2. $n_i(v) \neq 0 \implies (n_i(v), \lambda(v), N_i(v)) \in M_i(v)$,
3. $\forall (n,p,q) \in N_i(v), n \neq 0$ and $\exists (n, \ell', \mathcal{N}') \in M_i(v)$,
4. $\forall (n,p,q), (n',p',q') \in N_i(v), q \neq q'$,
5. $\forall (n(v_0), \ell, \mathcal{N}) \in M_i(v)$, $(\ell, \mathcal{N}) \preceq (\lambda(v_0), N(v_0))$,
6. $(n,p,q) \in N_i(v)$ if and only if the last message received by v through port q was $< (n, M), p >$ for some $M \subseteq M_i(v)$.

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

Lemma 4.2. For each vertex v and each step i, $n_i(v) \leq n_{i+1}(v)$, $N_i(v) \preceq N_{i+1}(v)$, and $M_i(v) \subseteq M_{i+1}(v)$.

Proof:
We suppose that some internal event is executed at step $i+1$ by some vertex $v \in V(G)$. The property is obviously true for any vertex $w \in V(G) \setminus \{v\}$ and it is easy to see that $M_i(v) \subseteq M_{i+1}(v)$.

If $n_i(v) \neq n_{i+1}(v)$, then $n_{i+1}(v) = 1 + \max\{n_1 \mid (n', \ell', \mathcal{N}') \in M_i(v)\}$ 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 $n_i(v) < n_{i+1}(v)$.

If $N_i(v) \neq N_{i+1}(v)$, then v has received a message $< (n', n'_{old}, M'), p >$ through port q and $N_{i+1}(v) = N_i(v) \setminus \{(n'_{old}, p, q)\} \cup \{(n', p, q)\}$. Let v' be the neighbour of v such that $\nu_v(v') = q$; we know that $\nu_{v'}(v) = p$.

If $(n'_{old}, p, q) \notin N_i(v)$, then $\max N_{i+1}(v) \triangle N_i(v) = (n', p, q) \in N_{i+1}(v)$ and then $N_i(v) \prec N_{+1}(v)$.

If $(n'_{old}, p, q) \in N_i(v)$, then $n'_{old} \neq n'$. Let $j < i+1$ be the computation step where v' has sent the message $< (n', n'_{old}, M'), p >$. We know that $n'_{old} \leq n' = n_j(v')$ and consequently, $\max N_{i+1}(v) \triangle N_i(v) = (n', p, q) \in N_{i+1}(v)$ and $N_i(v) \prec N_{+1}(v)$. □

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 exists $\ell, \mathcal{N}$ such that $(m, \ell, \mathcal{N}) \in M_i(v)$), 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 each vertex $v \in V(G)$, each step i and each $(m, \ell, \mathcal{N}) \in M_i(v)$, there exists a vertex $v' \in V(G)$ such that $n_i(v') = m$.

Proof:
We first note that $(m, \ell, \mathcal{N})$ is added to $\bigcup_{v \in V(G)} M_i(v)$ at some step i only if there exists a vertex $v \in V(G)$ such that $n_i(v) = m$, $\lambda(v) = \ell$ and $N_i(v) = \mathcal{N}$.

Given a vertex $v \in V(G)$, a step i and an element $(m, \ell, \mathcal{N}) \in M_i(v)$, let $U = \{(u, j) \in V(G) \times \mathbb{N} \mid j \leq i, n_j(u) = m\}$ and $U' = \{(u, j) \in U \mid \forall (u', j') \in U, (\lambda(u'), N_{j'}(u')) \prec (\lambda(u), N_j(u))$ or $(\lambda(u'), N_{j'}(u')) = (\lambda(u), N_j(u))$ and $j' \leq j\}$. Since $(m, \ell, \mathcal{N}) \in M_i(v)$, U and U' are both non-empty and it is easy to see that there exists i_0 such that for each $(u, j) \in U'$, $j = i_0$.

If $i_0 < i$, let $(u, i_0) \in U'$; we know that $n_{i_0+1}(u) \neq n_{i_0}(u)$, but this is impossible, since by maximality of $(\lambda(u), N_{i_0}(u))$, u cannot have modified its number. Consequently, $i_0 = i$ and there exists $v' \in V(G)$ such that $n_i(v') = m$. □

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 each vertex $v \in V(G)$ and each step i, for every $(m, \ell, \mathcal{N}) \in M_i(v)$, for every $m' \in [1, m]$, there exists $(m', \ell', \mathcal{N}') \in M_i(v)$.

Proof:
We prove this lemma by induction on i. Initially, the property is obviously true. We suppose that the property is satisfied at step i and that some vertex v executes some internal transition at step $i + 1$. If v executes the action **I** at step $i + 1$, then $M_i(v) = \{(1, \lambda(v), \emptyset)\}$ and the property is obviously true.

If v executes the action **R** at step $i+1$, then v has received some message $< (n', n'_{old}, M), p >$ from a neighbour v'. Let $j < i + 1$ be the computation step where v' has sent this message ; we know that $M' = M_j(v')$. If v does not modify its number at step $i + 1$, then $\{m \mid \exists (m, \ell, \mathcal{N}) \in M_{i+1}(v)\} = \{m \mid \exists (m, \ell, \mathcal{N}) \in M_i(v) \cup M_j(v')\}$ and the property is true at step $i + 1$ by the induction hypothesis. If v modifies its number at step $i + 1$, then $n_{i+1}(v) = 1 + \max\{m \mid \exists (m, \ell, \mathcal{N}) \in M_i(v) \cup M_j(v')\}$ and $M_{i+1}(v) = M_i(v) \cup M_j(v') \cup \{(n_{i+1}(v), \lambda(v), N_{i+1}(v)\}$. Then, the property is also satisfied at step $i + 1$ by the induction hypothesis. □

We now want to prove that any execution of $\mathcal{M}$ on $(\mathbf{G}, \nu)$ terminates. We know that a vertex does not send any message if its state $(\lambda(v), n(v), N(v), M(v))$ is not modified once it has executed some internal event. Then it is sufficient to prove that there exists a step i such that for each step $i' \geq i$ and for each vertex $v \in V(G)$, $n_{i'}(v) = n_i(v)$, $N_{i'}(v) = N_i(v)$ and $M_{i'}(v) = M_i(v)$. From Lemmas 4.3 and 4.4, we know that for each computation step i, the set $\{n_i(v) \mid v \in V(G)\}$ is a set $[0, k]$ or $[1, k]$ with $k \leq V(G)$. Then, from Lemma 4.2, we know that there exists a step i_0 such that for each vertex $v \in V(G)$ and for each step $i \geq i_0$, $n_i(v) = n_{i_0}(v)$. Consequently, we know that for each vertex v, $N_i(v)$ can only take a finite number of values and then, it is the same for $M_i(v)$. Consequently, from Lemma 4.2, we know that any execution ρ of $\mathcal{M}$ on $\mathbf{G}$ terminates.

We can then describe properties satisfied by the final configuration of any execution of the algorithm $\mathcal{M}$ on $\mathbf{G}$. First, we note that in the final configuration, there is no message in transit.

Lemma 4.5. Any execution ρ of $\mathcal{M}$ on a simple labelled graph $\mathbf{G} = (G, \lambda)$ with a port-numbering ν terminates and the final labelling $(\lambda, n_\rho, N_\rho, M_\rho)$ of the vertices of G satisfies the following properties:

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 $v, v' \in V(G)$:

2. $M_\rho(v) = M_\rho(v')$,
3. $(n_\rho(v), \lambda(v), N_\rho(v)) \in M_\rho(v')$,
4. if $n_\rho(v) = n_\rho(v')$, then $\lambda(v) = \lambda(v')$ and $N_\rho(v) = N_\rho(v')$,
5. $(n, p, q) \in N_\rho(v)$ if and only if there exists $w \in N_G(v)$ such that $\nu_v(w) = q$, $\nu_w(v) = p$ and $n_\rho(w) = n$.

Proof:

1. From Lemmas 4.3 and 4.4 and since any vertex has applied one of the actions $\mathbf{I}, \mathbf{R}$.
2. Since each time a vertex modifies its mailbox, it sends it to its neighbours and since all messages have arrived.
3. It follows from the previous property and from Lemma 4.1.
4. From Lemma 4.1.
5. From Lemma 4.1 and since all messages have arrived.

□

Thanks to Lemma 4.5, one can show that the final labelling of $(\mathbf{G}, \nu)$ enables to construct a digraph $(\mathbf{D}, \nu')$ such that $(Dir(\mathbf{G}), \nu)$ is a symmetric covering of $(\mathbf{D}, \nu')$.

Proposition 4.1. Given a graph $\mathbf{G}$ with a port numbering ν, we can associate with the final labelling of any execution ρ of $\mathcal{M}$ on $(\mathbf{G}, \nu)$, a digraph $(\mathbf{D}, \nu')$ such that $(Dir(\mathbf{G}), \nu)$ is a symmetric covering of $(\mathbf{D}, \nu')$.

Proof:
We use the notation of Lemma 4.5.

We consider the digraph D defined by $V(D) = \{m \in \mathbb{N} \mid \exists v \in V(G), n_\rho(v) = m\}$, $A(D) = \{a_{(n,p,q,m)} \mid \exists \{v, w\} \in E(G)$ such that $n_\rho(v) = n, n_\rho(w) = m, \delta_v(w) = p, \delta_w(v) = q\}$ and for each arc $a_{(n,p,q,m)}$, $s(a_{(n,p,q,m)}) = n$ and $t(a_{(n,p,q,m)}) = m$. We define the function $Sym : A(D) \to A(D)$ as follows: for each arc $a_{(n,p,q,m)} \in A(D)$, $Sym(a_{(n,p,q,m)}) = a_{(m,q,p,n)}$. Note that for each arc $a_{(n,p,p,n)}$, $Sym(a_{(n,p,p,n)}) = a_{(n,p,p,n)}$.

We define a labelling η of the vertices of D as follows. For each $v \in V(G)$, $\eta(n_\rho(v)) = \lambda(v)$. From Lemma 4.5, two vertices v, v' with the same final number have the same label $\lambda(v)$ and then this labelling is well-defined. We define the labelling ν' of the arcs of D as follows: for each arc $a_{(n,p,q,m)} \in A(D)$, $\nu'(a_{(n,p,q,m)}) = (p, q)$. Thus, for each arc $a_{(n,p,q,m)}$, $\nu'(Sym(a_{(n,p,q,m)})) = \nu'(a_{(m,q,p,n)}) = (q, p)$ and the arc labelling of D is symmetric.

We define an homomorphism φ from $(Dir(\mathbf{G}),\nu)$ to $(\mathbf{D},\nu)$ as follows. For each $v \in V(Dir(G))$, $\varphi(v) = n_\rho(v)$ and for each $a \in A(Dir(G))$, $\varphi(a) = a_{(n_\rho(s(a)),\nu_{s(a)}(t(a)),\nu_{t(a)}(s(a)),n_\rho(t(a)))}$. Since, for each arc $a \in A(Dir(\mathbf{G}))$, $s(\varphi(a)) = n_\rho(s(a)) = \varphi(s(a))$ and $t(\varphi(a)) = n_\rho(t(a)) = \varphi(t(a))$, φ is an homomorphism from $Dir(G)$ to D.

Since for each arc $a \in A(Dir(G))$, $\nu(a) = (\nu_{s(a)}(t(a)), \nu_{t(a)}(s(a))) = \nu'(\varphi(a))$ and since for each vertex $v \in V(Dir(G))$, $\lambda(v) = \eta(\varphi(v))$, φ is a homomorphism from $(Dir(\mathbf{G}),\nu)$ to $(\mathbf{D},\nu')$. Moreover, for each arc $a \in A(Dir(G))$, $\varphi(Sym(a)) = a_{(n_\rho(t(a)),\nu_{t(a)}(s(a)),\nu_{s(a)}(t(a)),n_\rho(s(a)))} = Sym(\varphi(a))$.

For each arc $a_{(n,p,q,m)} \in A(D)$, there exists an edge $\{v,w\} \in E(G)$ such that $n_\rho(v) = n$, $\nu_v(w) = q$, $\nu_w(v) = p$ and $n_\rho(w) = m$. From Lemma 4.5, we know that for each $u \in \varphi^{-1}(n)$ (resp. $u \in \varphi^{-1}(m)$), $N_\rho(u) = N_\rho(v)$ (resp. $N_\rho(u) = N_\rho(w)$) and therefore, since ν is a bijection between $N_G(u)$ and $[1, \deg_G(u)]$, there exists a unique $u' \in N_G(u)$ such that $n_\rho(u') = m$ (resp. $n_\rho(u') = n$), $\nu_u(u') = p$ (resp. $\nu_u(u') = q$) and $\nu_{u'}(u) = q$ (resp. $\nu_{u'}(u) = p$). Thus, for each arc $a_{(n,p,q,m)} \in A(D)$, for each vertex $u \in \varphi^{-1}(s(a))$ (resp. $u \in \varphi^{-1}(t(a))$), there exists a unique arc $a \in Dir(G)$ such that $\varphi(a) = a_{(n,p,q,m)}$ and $s(a) = u$ (resp. $t(a) = u$). Consequently, $(Dir(\mathbf{G}),\nu)$ is a symmetric covering of $(\mathbf{D},\nu')$. □

Consider now a graph $\mathbf{G}$ with a port-numbering ν such that $(Dir(\mathbf{G}),\nu)$ is symmetric covering prime. For every execution ρ of $\mathcal{M}$ on $(\mathbf{G},\nu)$, the digraph obtained from the final labelling is isomorphic to $(Dir(\mathbf{G}),\nu)$ and therefore the set of numbers of the vertices is exactly $[1,|V(G)|]$: each vertex has a unique number. Moreover, the termination detection of the algorithm is possible on $\mathbf{G}$. Indeed, once a vertex gets the identity number $|V(G)|$, from Lemmas 4.3 and 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. In this case, one can also solve the election problem, since this vertex can take the label *elected* and broadcasts the information that a vertex has been elected.

Furthermore, it has been shown in Proposition 3.2 that for every graph $\mathbf{G}$ with a port-numbering ν such that $(Dir(\mathbf{G}),\nu)$ is not symmetric covering prime, there does not exist any algorithm that solves the naming problem or the election problem on $(\mathbf{G},\nu)$. Thus we have proven the following theorem.

Theorem 4.1. Given a simple labelled graph $\mathbf{G}$ with a port-numbering ν, there exists an election (or a naming) algorithm for $(\mathbf{G},\nu)$ if and only if $(Dir(\mathbf{G}),\nu)$ is symmetric covering prime.

Remark 4.1. Note that any execution of $\mathcal{M}$ on a network always terminates, even if the vertices cannot detect that the computation is over if they have no information about the network.

To detect the termination of the execution of $\mathcal{M}$ on some graph $\mathbf{G}$ with a port-numbering ν, the vertices of $\mathbf{G}$ do not need to know the topology of $\mathbf{G}$: the knowledge of its size is sufficient.

As Yamashita and Kameda, we are interested in characterizing graphs that admit an election (or a naming) algorithm for any port-numbering. We present this characterization in the next theorem.

Theorem 4.2. Given a simple labelled graph $\mathbf{G}$, there exists an election (or a naming) algorithm for $\mathbf{G}$ if and only if $Dir(\mathbf{G})$ is symmetric covering prime.

Proof:
It is easy to see that if $Dir(\mathbf{G})$ is symmetric covering prime, then for any port-numbering ν of $\mathbf{G}$,

$(Dir(\mathbf{G}), \nu)$ is symmetric covering prime and consequently, there exists an election (or a naming) algorithm for $\mathbf{G}$.

We now show that if $Dir(\mathbf{G})$ is a proper symmetric covering of some symmetric digraph $\mathbf{D}$ via a homomorphism φ, then there exists a symmetric labelling ν' of the arcs of $\mathbf{D}$ and a port-numbering ν of $\mathbf{G}$ such that $(Dir(\mathbf{G}), \nu)$ is a symmetric covering of $(\mathbf{D}, \nu')$. In this case, there exists a port-numbering ν of $\mathbf{G}$ such that it is impossible to solve the election (or the naming) on $(\mathbf{G}, \nu)$.

We first give a number $p(a)$ to each arc $a \in A(D)$ such that for each vertex v, p induces a bijection between the outgoing arcs of v and $[1, \deg^+(v)]$ where $\deg^+(v)$ is the number of outgoing arcs of v in D. Then we define the labelling ν' of the arcs of D as follows: for each arc $a \in A(D)$, $\nu'(a) = (p(a), p(Sym(a)))$. It is easy to see that the labelling ν' is a symmetric labelling of the arcs where for each $(p, q) \in \mathbb{N}^2$, $\iota((p, q)) = (q, p)$.

In order to define the port-numbering ν of $\mathbf{G}$, we proceed as follows. For each edge $\{u, v\} \in E(G)$, we define $\nu_u(v) = p(\varphi(a_{(u,v)}))$ and $\nu_v(u) = p(\varphi(a_{(v,u)}))$. Since φ is a covering and from the definition of p, it is clear that $\nu = \{\nu_u \mid u \in V(G)\}$ is a port-numbering.

It remains to show that $(Dir(\mathbf{G}), \nu)$ is a symmetric covering of $(\mathbf{D}, \nu')$ via φ, i.e., that φ preserves the labels of the arcs. For each arc $a_{(u,v)} \in A(Dir(G))$, $\nu(\varphi(a_{(u,v)}) = (\nu_u(v), \nu_v(u)) = (p(\varphi(a_{(u,v)})), p(\varphi(a_{(v,u)}))) = (p(\varphi(a_{(u,v)})), p(\varphi(Sym(a_{(u,v)})))) = (p(\varphi(a_{(u,v)})), p(Sym(\varphi(a_{(u,v)}))))$ $= \nu'(\varphi(a_{(u,v)}))$. Consequently, $(Dir(\mathbf{G}), \nu)$ is a symmetric covering of $(\mathbf{D}, \nu')$ via φ. □

4.5. Remarks on the Algorithm $\mathcal{M}$

Contrary to the algorithms of Yamashita and Kameda [YK96] and of Boldi *et al.* [BCG+96], the algorithm $\mathcal{M}$ is totally asynchronous, in the sense that, once a vertex has sent messages to all its neighbours, it does not have to wait for a message from each of its neighbours in order to continue to execute the algorithm. An interesting consequence of this property is that for any network $(\mathbf{G}, \nu)$, there exists an execution of $\mathcal{M}$ on $(\mathbf{G}, \nu)$ that enables to elect a vertex, even if $(Dir(\mathbf{G}), \nu)$ is not symmetric covering prime.

Proposition 4.2. For any network $(\mathbf{G}, \nu)$, there exists an execution of $\mathcal{M}$ on $(\mathbf{G}, \nu)$ that enables to solve naming and election on $(\mathbf{G}, \nu)$.

Proof:
Given a network $(\mathbf{G}, \nu)$, let us consider an execution where there is exactly one process v_0 that executes the action $\mathbf{I}$ and where all the other processes starts executing the algorithm only when they receive a message. In other words, all the processes are initially passive and there is a unique initiator v_0. Then, for any vertex $v \in V(G) \setminus \{v_0\}$, each time v modifies its number, there exists an element $(1, \lambda(v_0), \emptyset)$ in its mailbox and thus $n(v)$ is always different from 1. Consequently, if we consider the final labelling of $(\mathbf{G}, \nu)$ at the end of the execution, there exists a unique vertex v_0 whose number is 1. Consequently, from Propositions 2.1 and 4.1, we know that each number $n(v)$ that appears in the final labelling is the number of a unique vertex: each vertex has a unique number. Since, in this case, it is possible to detect the termination of the algorithm by Remark 4.1, we know that this execution enables to solve naming and election on $(\mathbf{G}, \nu)$. □

Nevertheless, for any network $(\mathbf{G}, \nu)$, there exists a "canonical" execution of $\mathcal{M}$ that solves naming and election on $(\mathbf{G}, \nu)$ if and only if $(Dir(\mathbf{G}), \nu)$ is symmetric covering prime.

A *synchronous* execution is an execution in *rounds*. At each round, each process receives the messages that have been sent to it during the previous round; then, it modifies its state and it sent as many messages as needed to its neighbours. Note that in a synchronous execution of $\mathcal{M}$ on a network $(\mathbf{G}, \nu)$, all the processes initially executes the action $\mathbf{I}$.

The following proposition is similar to impossibility results given by Yamashita and Kameda [YK96] and Boldi *et al.* [BCG$^+$96].

Proposition 4.3. Consider a network $(\mathbf{G}, \nu)$ and a synchronous execution of $\mathcal{M}$ on $(\mathbf{G}, \nu)$. The labelled digraph obtained from the final labelling is the minimum base of $(Dir(\mathbf{G}), \nu)$.

Proof:
Consider a network $(\mathbf{G}, \nu)$ and let $(\mathbf{D}, \nu')$ be the minimum base of $(\mathbf{G}, \nu)$. We denote by φ the fibration from $(Dir(\mathbf{G}), \nu)$ to $(\mathbf{D}, \nu')$. Note that $(Dir(\mathbf{G}), \nu)$ is a symmetric covering of $(\mathbf{D}, \nu')$ via φ (this is ensured by the fact that ν is a port-numbering of $\mathbf{G}$).

Let $v, v' \in V(G)$ be such that $\varphi(v) = \varphi(v')$. Initially, v and v' have the same label, they both apply the action $\mathbf{I}$ and they send the same messages to their neighbours. Moreover, for any $u \in N_G(v)$, there exists a unique vertex $u' \in N_G(v')$ such that $\nu_{v'}(u') = \nu_v(u)$, $\nu_{u'}(v') = \nu_u(v)$ and $\varphi(u') = \varphi(u)$. Consequently, for any round i, the vertices v and v' get exactly the same messages from their neighbours, modify their states in the same way and send the same messages to their neighbours.

Consequently, in the final configuration, for all vertices $v, v' \in V(G)$ such that $\varphi(v) = \varphi(v')$, $n(v) = n(v')$ and $N(v) = N(v')$. Consequently, the graph obtained from the final labelling is $(\mathbf{D}, \nu')$. □

We can remark that the mailbox of each vertex contains a lot of useless information. Indeed, if some $(n, \ell, \mathcal{N})$ belongs to the mailbox $M(v)$ of a vertex v, one can remove from $M(v)$ all the elements $(n, \ell', \mathcal{N}') \in M(v)$ such that $(\ell', \mathcal{N}') \prec (\ell, \mathcal{N})$. We can thus replace the mailbox $M(v)$ of v by $\{(n, \ell, \mathcal{N}) \in M(v) \mid \forall (n, \ell', \mathcal{N}') \in M(v), (\ell', \mathcal{N}') \preceq (\ell, \mathcal{N})\}$. In this way, the mailbox of each vertex contains at most $|V(G)|$ elements of the form $(n, \ell, \mathcal{N})$.

Moreover, it is not necessary to ensure that the communication channels preserve the order of messages (i.e., they have the FIFO property). Indeed, each time a vertex receives a message $< (n', n'_{old}, M'), p >$ through port q from a neighbour v', if there exists $(n, p, q) \in N(v)$ such that $n > n'$, then we know that v has received earlier a message $< (n, n_{old}, M), p >$ through port q that contains a more recent information about the state of v'. Moreover, we know that in this case, $M' \subseteq M$. Consequently, each time v receives a message $< (n', n'_{old}, M'), p >$ through port q, it checks if there exists $(n, p, q) \in N(v)$ and performs the following instructions. If $n > n'$, it just discards the message. Otherwise, if there is no such an element or if $n \leq n'$ then it updates its local view by removing (n, p, q) from $N(v)$ and adding (n', p, q) to $N(v)$ and it continues the algorithm as before.

Nevertheless, if we assume that the communication channels have the FIFO property, one can reduce the size of the messages. Indeed, each time a vertex modifies its mailbox, it just has to send the elements it adds to its mailbox instead of sending the whole mailbox. In the complexity analysis we do in the next section, we suppose that the communication channels have the FIFO property and that the sizes of messages and mailboxes are reduced as we just explained.

4.6. Complexity Analysis

We are interested in determining the time complexity, the message complexity of $\mathcal{M}$ and we want to determine the size of the messages and the size of the memory needed by each vertex.

As Tel [Tel00] (p. 71), we define the time complexity by supposing that internal events need zero time units and that the transmission time (i.e., the time between sending and receiving a message) is at most one time unit. This corresponds to the number of rounds needed by a synchronous execution of the algorithm. Note that the correctness of $\mathcal{M}$ is independent of these assumptions.

We consider an initial labelling λ of G such that the size of each label ℓ is at most $O(\log |V(G)|)$ bits (which is enough to give distinct labels to all vertices of G if needed).

We consider an execution of the algorithm where the size of messages and mailboxes has been reduced as explained in the previous section. Moreover, we suppose that each process sends the elements $\{(n', n'_{old}, \mathcal{N}')\}$ of its mailbox one by one and it sends k messages if it has k elements to send to its neighbours.

Proposition 4.4. Given a network $(\mathbf{G}, \nu)$ with n vertices, m edges, whose maximum degree is Δ and whose diameter is D, any execution of $\mathcal{M}$ on $(\mathbf{G}, \nu)$ needs $O(Dn^2)$ time units and $O(m^2 n)$ messages of $O(\Delta \log n)$ bits. Moreover, the memory needed by each process is $O(\Delta n \log n)$ bits.

Proof:
Consider a network $(\mathbf{G}, \nu)$ with n vertices, m edges, whose maximum degree is Δ and whose diameter is D. Consider an execution of $\mathcal{M}$ on $(\mathbf{G}, \nu)$. From Lemmas 4.3 and 4.4, we know that each vertex modifies its number at most n times.

For each vertex v, since the numbers of v and of its neighbours can only increase, the couple $(n(v), N(v))$ can take at most $(\deg_G(v) + 1)n$ different values. Each time a vertex modifies its number or its local view, it can generate at most $O(m)$ messages, since a vertex whose mailbox already contains $(n(v), \lambda(v), N(v))$ will not broadcast this information to its neighbours. Consequently, any execution of $\mathcal{M}$ on $\mathbf{G}$ needs at most $O(m^2 n)$ messages. Moreover, since we suppose that all messages have the form $< (n, n_{old}, \{(n', \ell', \mathcal{N}')\}), p >$ and since $N(v)$ contains at most Δ elements, each message has a size of $O(\Delta \log n)$ bits.

Each time a vertex v modifies its number or its local view, all vertices of G know $(n(v), \lambda(v), N(v))$ in D time units. Moreover, each time a vertex v modifies its number, the neighbours of v modify their local views in 1 time unit. Thus, each time a vertex modifies its number, all the vertices of G have the same mailbox in $D + 1$ time units if no vertex has modified its number in the meanwhile. If all vertices have the same mailbox and if all send messages have arrived, we know from Lemma 4.1, that the computation is over. Consequently, any execution of the algorithm terminates in $O(Dn^2)$ time units.

For each vertex v, $n(v)$ can be encoded with $\log n$ bits and $N(v)$ can be encoded with $O(\Delta \log n)$ bits. Since each vertex keeps only the information that is useful in its mailbox, there are at most n elements in $M(v)$ and each of these elements can be encoded with $O(\Delta \log n)$ bits. Consequently, each vertex of G needs a memory of $O(\Delta n \log n)$ bits to store its state. □

Remark 4.2. For any network $(\mathbf{G}, \nu)$, the algorithms of Yamashita and Kameda [YK96] and of Boldi *et al.* [BCG$^+$96] needs $O(n)$ time units and $O(mn)$ messages of $2^{O(n)}$ bits. Moreover, each process needs $2^{O(n)}$ bits of memory.

Thus, $\mathcal{M}$ is an algorithm that is executed in polynomial time and needs messages of polynomial size, whereas the algorithms known before need messages of exponential size.

5. Final Remarks

In this paper, we have presented a characterization of networks that admit naming and election algorithms in the asynchronous message passing model. This characterization is expressed in terms of coverings of directed graphs. To obtain a sufficient condition, we have adapted Mazurkiewicz's algorithm in the asynchronous message passing model and the enumeration algorithm we obtained has some nice properties that the previous existing algorithms do not have. In particular, our algorithm needs a polynomial number of messages of polynomial size, whereas existing algorithms use messages of exponential size.

In [YK99], Yamashita and Kameda consider different models of port awareness. In the algorithm they use to solve the election problem, if a vertex sends a message, it sends it to all its neighbours. The algorithm we described in Section 4 uses the same technique. Therefore they consider two kinds of emissions of messages: either each vertex can distinguish its outgoing ports or not. If it cannot, then each vertex can just broadcast a message to all its neighbours. Moreover they also distinguish two kinds of receptions of messages: either a vertex can know the port through which it receives each message, or it receives all messages in a mailbox and it has no means to know who is the sender of each message. Consequently, in [YK99], four message passing models are studied: the *port-to-port* model (that has been studied here), the *port-to-mailbox* model, the *broadcast-to-port* model and the *broadcast-to-mailbox* model. One can also obtain characterizations of networks admitting election and naming algorithms in these different models expressed in terms of coverings and fibrations. Moreover, one can easily adapt algorithm $\mathcal{M}$ to these different models to obtain polynomial algorithms, whereas the algorithms presented by Yamashita and Kameda in [YK99] need messages of exponential size.

In Mazurkiewicz's model, the algorithm presented by Mazurkiewicz in [Maz97] is used as a basic building block to solve other classical problems in distributed computing. In [GMM04], Godard, Métivier and Muscholl characterize graph classes that can be recognized in a distributed way. In [MT00], Métivier and Tel characterize graph classes where any distributed algorithm can be transformed into a distributed algorithm where processes can detect that the global computation is finished and in [GM02], Godard and Métivier characterize graph classes that admit an election algorithm.

It is a natural question to wonder if these results can be extended to the asynchronous message passing model, once we have a Mazurkiewicz-like algorithm in this model. In [CGMT07], using the algorithm described in this paper and a termination detection algorithm of Szymansky, Shi and Prywes [SSP85], some extensions of the results of [MT00, GM02] have been obtained in the asynchronous message passing model.

References

[Ang80] D. Angluin. Local and global properties in networks of processors. In *Proc. of the 12th Symposium on Theory of Computing (STOC 1980)*, pages 82–93, 1980.

[AW04] H. Attiya and J. Welch. *Distributed computing: fundamentals, simulations, and advanced topics*. John Wiley and Sons, 2004.

[BCG+96] P. Boldi, B. Codenotti, P. Gemmell, S. Shammah, J. Simon, and S. Vigna. Symmetry breaking in anonymous networks: characterizations. In *Proc. of the 4th Israeli Symposium on Theory of Computing and Systems (ISTCS 1996)*, pages 16–26. IEEE Press, 1996.

[Bod89] H.L. Bodlaender. The classification of coverings of processor networks. *Journal of parallel and distributed computing*, 6(1):166–182, 1989.

[BV02] P. Boldi and S. Vigna. Fibrations of graphs. *Discrete Mathematics*, 243(1-3):21–66, 2002.

[CGMT07] J. Chalopin, E. Godard, Y. Métivier, and G. Tel. About the termination detection in the asynchronous message passing model. In *Proc. of the 33rd Conference on Current Trends in Theory and Practice of Informatics (SOFSEM 2007)*, volume 4362 of *Lecture Notes in Computer Science*, pages 200–211. Springer-Verlag, 2007.

[Cha05] J. Chalopin. Local computations on closed unlabelled edges: the election problem and the naming problem. In *Proc. of the 31st Conference on Current Trends in Theory and Practice of Informatics (SOFSEM 2005)*, volume 3381 of *Lecture Notes in Computer Science*, pages 81–90. Springer-Verlag, 2005.

[CM04] J. Chalopin and Y. Métivier. Election and local computations on edges. In *Proc. of Foundations of Software Science and Computation Structures, 7th International Conference (FOSSACS 2004)*, volume 2987 of *Lecture Notes in Computer Science*, pages 90–104. Springer-Verlag, 2004.

[CM05] J. Chalopin and Y. Métivier. A bridge between the asynchronous message passing model and local computations in graphs. In *Proc. of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS 2005)*, volume 3618 of *Lecture Notes in Computer Science*, pages 212–223. Springer-Verlag, 2005.

[CMZ04] J. Chalopin, Y. Métivier, and W. Zielonka. Election, naming and cellular edge local computations. In *Proc. of the 2nd International Conference on Graph Transformations (ICGT 2004)*, volume 3256 of *Lecture Notes in Computer Science*, pages 242–256. Springer-Verlag, 2004.

[GM02] E. Godard and Y. Métivier. A characterization of families of graphs in which election is possible (*ext. abstract*). In *Proc. of Foundations of Software Science and Computation Structures, 5th International Conference (FOSSACS 2002)*, volume 2303 of *Lecture Notes in Computer Science*, pages 159–172. Springer-Verlag, 2002.

[GMM04] E. Godard, Y. Métivier, and A. Muscholl. Characterization of classes of graphs recognizable by local computations. *Theory of Computing Systems*, 37(2):249–293, 2004.

[HU79] J. Hopcroft and J. Ullman. *Introduction to automata theory, languages and computation*. Addison-Wesley, 1979.

[Lei82] F.T. Leighton. Finite common coverings of graphs. *J. Combin. Theory, Ser. B*, 33(3):231–238, 1982.

[LeL77] G. LeLann. Distributed systems, towards a formal approach. In *Information processing 1977*, pages 155–160. North-Holland, 1977.

[Lyn96] N. Lynch. *Distributed algorithms*. Morgan Kaufman, 1996.

[Maz97] A. Mazurkiewicz. Distributed enumeration. *Information Processing Letters*, 61(5):233–239, 1997.

[Maz04] A. Mazurkiewicz. Bilateral ranking negotiations. *Fundamenta Informaticae*, 60(1-4):1–16, 2004.

[MT00] Y. Métivier and G. Tel. Termination detection and universal graph reconstruction. In *Proc. of 7th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2000)*, Proceedings in Informatics, pages 237–251. Carleton Scientific, 2000.

[Nor95] N. Norris. Universal covers of graphs: isomorphism to depth $n-1$ implies isomorphism to all depths. *Discrete Applied Math.*, 56(1):61–74, 1995.

[SSP85] B. Szymanski, Y. Shy, and N. Prywes. Terminating iterative solutions of simultaneous equations in distributed message passing systems. In *Proc. of the 4th Annual ACM Symposium on Principles of Distributed Computing (PODC 1985)*, pages 287–292. ACM Press, 1985.

[Tel00] G. Tel. *Introduction to distributed algorithms*. Cambridge University Press, 2000.

[TvS02] A. Tanenbaum and M. van Steen. *Distributed Systems - Principles and Paradigms*. Prentice Hall, 2002.

[YK96] M. Yamashita and T. Kameda. Computing on anonymous networks: Part i - characterizing the solvable cases. *IEEE Transactions on parallel and distributed systems*, 7(1):69–89, 1996.

[YK99] M. Yamashita and T. Kameda. Leader election problem on networks in which processor identity numbers are not distinct. *IEEE Transactions on parallel and distributed systems*, 10(9):878–887, 1999.