Election and Rendezvous with Incomparable Labels*

Jérémie Chalopin

chalopin@labri.fr
LaBRI, Université Bordeaux 1
351 cours de la Libération
33405 Talence, France

Abstract. In *"Can we elect if we cannot compare" (SPAA'03)*, Barrière, Flocchini, Fraigniaud and Santoro consider a *qualitative* model of distributed computing, where the labels of the entities are distinct but mutually incomparable. They study the leader election problem in a distributed mobile environment and they wonder whether there exists an algorithm such that for each distributed mobile environment, it either states that the problem cannot be solved in this environment, or it successfully elects a leader. In this paper, we give a positive answer to this question. We also give a characterization of the distributed mobile environments where election and rendezvous can be solved.

1 Introduction

Consider an intercontinental highway network linking different cities in different countries. In each city, the directions to the other cities are written in the language that is locally spoken. Consider now a set of different drivers coming from different countries. Initially, each driver starts in his town and all the drivers want to meet at a single place. The only mean they have to communicate is to leave messages in each city they reach, but each driver can only speak his mother tongue: he can see that another driver left some message, but he cannot understand it. Moreover, each driver can consistently distinguish the different directions in each city, but the drivers cannot agree on an alphabetical order on these directions: a French driver would not be able to figure out how to order Chinese words in the Chinese way, for example. We wonder whether there exists a procedure that enables them to meet at a single point in a finite time.

In distributed computing, the links incident to each process are usually labelled by distinct numbers in order to allow each process (or each mobile agent) to consistently distinguish its neighbours; this labelling is usually called a port-numbering. In fact, these numbers enable not only to distinguish the links, but also to order them. Many distributed algorithms assume also that all the processes can be unambiguously identified, and therefore the processes are given numbers. Again, one can see that this enables to order the different processes according to their labels. This usual setting is a *quantitative* model, since each label can be seen as a number.

Nevertheless, as in the example presented above, one may be able to distinguish labels without being able to order them. In this paper, we consider distributed mobile environments where mobile agents are scattered all over a network. All the agents have distinct *colors* (their labels), which are mutually incomparable: each agent can just check whether two colors are equal or not. The links incident to each vertex are also given distinct incomparable colors. This model is close to the one introduced by Barrière *et al.* in [BFFS03]; it is *qualitative*, in the sense that there is no a priori order between the labels. As in [BFFS03], we study the impact of the lack of a total order on the set of labels in a distributed mobile environment. In this way, we investigate the leader election problem, that is a classical problem to highlight the differences between various models of distributed computing.

In usual models, there is always an implicit order over the set of labels, since for each agent, each information is just a sequence of bits. Nevertheless, consider an algorithm designed to be executed by mobile agents over a network. If the agents have been implemented by different companies, and if the specifications of the algorithm do not specify how the integers must be represented, some agents can for instance store

* A preliminary version of this paper has been published in [Cha06]

numbers with most significant bit first whereas other agents store numbers with least significant bit first; in this case, the agents would not agree on the meaning of the sequence 01101. Moreover, it is always interesting to deal with algorithms that need less specifications, since they are generally more robust, and easier to implement in different models of distributed computing.

1.1 The Model

In this paper, an agent is an entity which executes an algorithm: it can move from place to place (with some data and its algorithm) through communication links, it can make local computations on a place (a place provides tools for local computations: data, memories and process) and leave messages on a place.

In our model, the environment is represented by a simple undirected connected graph $G = (V(G), E(G))$ and a set $\mathcal{E}$ of mobile agents is scattered over G. We suppose that the graph G is anonymous, i.e., vertices of G do not have identities. Communications between agents is achieved through writing messages on *whiteboards*, where agents can read, write, and erase messages. There is one whiteboard on each vertex of G, and access to a whiteboard is in fair mutual exclusion.

Each agent $r \in \mathcal{E}$ is initially placed on a vertex of the graph and we denote by p the injection describing the initial placement of the agents in G. The vertex $p(r)$ is called the *homebase* of the agent $r \in \mathcal{E}$ and we suppose that initially, the homebases are marked: they contain a marker that enables each agent to know that a place is a homebase. We will denote such a distributed mobile environment by $(G, \mathcal{E}, p)$.

We consider an injective function *color* that associates to each agent $r \in \mathcal{E}$ a unique color from a set C_a. There is no *a priori* order on the set of colors: each agent can give its own order on the set of colors, but the agents do not necessarily agree on a particular order. If there is a message on some whiteboard, an agent can always know the color of the message (i.e., the color of its author), but it cannot understand it if the color of the message is different from its own color (i.e., it can only understand the messages it has written). We also suppose that each time an agent r is on the homebase $p(r')$ of an agent r', it can detect the color of this homebase (i.e., the color of r').

In each place, the incident links are labelled by different colors that enable each agent to consistently distinguish the neighbours of the place: for each vertex u, there exists an injective function δ_u that associates a color from a set C_δ (disjoint from C_a) to each edge incident to u. The set $\delta = \{\delta_u : u \in V(G)\}$ constitutes the *port-labelling* of G. Thanks to this labelling δ, each agent can make a distinction between the incident edges of each vertex. As for C_a, the agents do not agree on an *a priori* order on the set of colors C_δ. Such a distributed colored mobile environment will be denoted by $(G, \delta, \mathcal{E}, p, color)$.

The agents are asynchronous, in the sense that every action they perform (computing, moving, etc.) takes a finite but otherwise unpredictable amount of time. The actions an agent a located at a node v can perform depends on the current state of a, the current state of the whiteboard at v, and the color of the port through which a entered v. According to these information, a can decide to write a message on the whiteboard of v, to leave v (through a port whose color may result from some computation), or to stay at v (for example, to wait that another agent leaves a message on the whiteboard).

As in [BFFS03], we suppose that an agent has not an initial knowledge of the network topology, neither of its size nor of the number of agents in the system. Nevertheless, the model considered in this paper is more restrictive than the one presented in [BFFS03], since in the model of Barrière *et al.* the agents cannot agree on an order on the set of colors, but they fully understand the symbols written by the other agents. However, the necessary condition presented in our model is the same as the one presented in [BFFS03]: the results presented in this paper remain true in the model of [BFFS03].

1.2 Election and Rendezvous

The election problem is one of the paradigms of the theory of distributed computing, that was first posed by LeLann [LeL77]. In the distributed mobile setting, the aim of a leader election algorithm is to distinguish one agent among the others. All the agents execute the same protocol, i.e., the only initial difference between two agents is their colors. At the end of the execution of the algorithm, there is exactly one agent in the

state *elected*, whereas all the other agents enter the state *non-elected*. Moreover, it is supposed that once an agent enters the state *elected* or *non-elected*, it remains in such a state until the end of the computation.

Another important problem in this setting is the *rendezvous* problem. The aim of a rendezvous algorithm is to reach a configuration where all the mobile agents gather in the same place. These two problems are equivalent, since once an agent has been elected, if all the agents agree on the label *elected*, all the agents can gather in the homebase of the elected agent. Reversely, once all the agents have gathered in some place, the first agent that writes on the whiteboard of this place is elected, whereas all the others become non-elected. There exists a large variety of results for these problems in the mobile agent setting assuming different properties of the environment [BFFS06,DFNS05,DFP03,KKR06]. The election problem has also been extensively studied in the distributed setting, and particularly in anonymous networks, where the processes do not have distinct labels [Ang80,BCG$^+$96,CM05,YK96,YK99].

In the model we consider, an election (or rendezvous) algorithm for a distributed mobile environment must not use some particularity of the colors used to label the ports or the agents or make any assumption on the set of colors (for example, one cannot design an algorithm that elects a red agent when there is such an agent). Consider a graph G and a set of agents $\mathcal{E}$ scattered over the network according to a function p. We say that we can solve the election (resp. rendezvous) problem on $(G, \mathcal{E}, p)$ if the problem can be solved on $(G, \delta, \mathcal{E}, p, color)$ for all port-labellings δ and all agent-coloring functions $color$. Note that, as for anonymous networks in the distributed setting [CM05,YK96], the protocols must not depend on the port-labelling. Indeed, the role of the port-labelling is just to enable an agent to make a distinction between the different neighbours of a vertex.

As in [BFFS03], we say that an algorithm $\mathcal{A}$ is an *effective* election (resp. rendezvous) algorithm if for each distributed mobile environment $(G, \mathcal{E}, p)$, each port-labelling δ and each coloring function $color$, for all the executions of $\mathcal{A}$ on $(G, \delta, \mathcal{E}, p, color)$, either all the agents detect that the election (resp. rendezvous) problem cannot be solved in $(G, \mathcal{E}, p)$ (i.e., there does not exist any algorithm that can solve the problem in this environment), or the agents successfully elect one of them (resp. gather in some vertex). In particular, note that such an algorithm does not need any initial knowledge about the topology, the size, the diameter of the network or about the number of agents.

In the following, we will mainly focus on the election problem, but we have to keep in mind the results obtained remain true for the rendezvous problem.

1.3 Main Results

In this work, we give a characterization (Theorem 1) of distributed mobile environments, where the election problem can be solved.

In [BFFS03], Barrière *et al.* wonder whether there exists an effective election algorithm for the qualitative world. The algorithm we describe gives a positive answer to this question (Theorem 2).

To obtain a necessary condition (Proposition 3), we use *well-balanced* automorphisms that have been introduced by Bougé in [Bou88].

Then, we show that this necessary condition is also sufficient: we use some links between fibrations and automorphisms presented in [BV02] to describe an effective algorithm in Section 4.2 that solves the election problem when the necessary condition is satisfied.

2 Preliminaries

2.1 Labelled Digraphs

In the following, we will consider directed graphs (digraphs) with multiple arcs and self-loops. A *digraph* $D = (V(D), A(D), s_D, t_D)$ is defined by a set $V(D)$ of vertices, a set $A(D)$ of arcs and by two maps s_D and t_D that assign to each arc two elements of $V(D)$: a source and a target (in general, the subscripts will be omitted); if a is an arc, the arc a is said to be going out of $s(a)$ and coming into $t(a)$. We say that $s(a)$ is a predecessor of $t(a)$ and that $t(a)$ is a successor of $s(a)$.

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 $Sym : A(D) \to A(D)$ such that for every $a \in A(D), s(a) = t(Sym(a))$.

Definition 1. *A* homomorphism γ *between the digraph D and the digraph D' is a mapping $\gamma\colon V(D) \cup A(D) \to V(D') \cup A(D')$ such that for each arc $a \in A(D)$, $\gamma(s(a)) = s(\gamma(a))$ et $\gamma(t(a)) = t(\gamma(a))$.*

An homomorphism γ is an isomorphism *if γ is bijective.*

Throughout the paper we will consider digraphs where the vertices and the arcs are labelled with labels from a recursive label set L. A digraph G 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 $\gamma\colon V(D) \cup A(D) \to V(D') \cup A(D')$ is a homomorphism from (D, λ) to (D', λ') if γ is a digraph homomorphism from D to D' which preserves the labelling, i.e., such that $\lambda'(\gamma(x)) = \lambda(x)$ for every $x \in V(D) \cup A(D)$. Labelled digraphs will be designated by bold letters like $\mathbf{D}, \mathbf{G}, \dots$ If $\mathbf{D}$ is a labelled digraph, then D denotes the underlying digraph.

Given a connected simple graph $G = (V(G), E(G))$, one associate a symmetric strongly connected digraph denoted by $Dir(G)$ and defined as follows: $V(Dir(G)) = V(G)$ and for each edge $\{u, v\} \in E(G)$, there exist 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.

Given a mobile environment $(G, \mathcal{E}, p)$, we define the labelling function χ_p of the vertices by $\chi_p(v) = 1$ if there exists an agent a such that $p(a) = v$, and $\chi_p(v) = 0$ otherwise. A distributed mobile environment $(G, \mathcal{E}, p)$ can therefore be represented by the labelled digraph $(Dir(G), \chi_p)$.

2.2 Fibrations and Coverings

The notions of fibrations and coverings are fundamental in this work; definitions and main properties are presented in [BV02].

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

Definition 2. *A digraph D is* fibred over *a digraph D' via a homomorphism φ if φ is a homomorphism from D to D' such that 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 $\varphi(a) = a'$.*

The homomorphism φ is then called a fibration *from D to D' and the digraph D' is the* base *of φ.*

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

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

Definition 3. *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 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 then called a covering projection *from D to D'.*

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

Definition 4. *A symmetric digraph D is 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 a symmetric covering projection *from D to D'.*

A digraph D is symmetric-covering-minimal *if there does not exist any digraph D' not isomorphic to D such that D is a symmetric covering of D'.*

An interesting property satisfied by any covering projection is that the preimage of all the vertices of the base have the same size.

Proposition 1 ([BV02]). *Given two strongly connected digraphs D and D' and a covering projection φ from D to D', there exists $q \in \mathbb{N}$ such that for each $x' \in V(D') \cup A(D')$, $|\varphi^{-1}(x')| = q$.*

The notions of fibrations and coverings extend to labelled digraphs in a natural way: the homomorphisms must preserve the labelling. Examples of fibrations and coverings are given in Figures 1, 2 and 4.

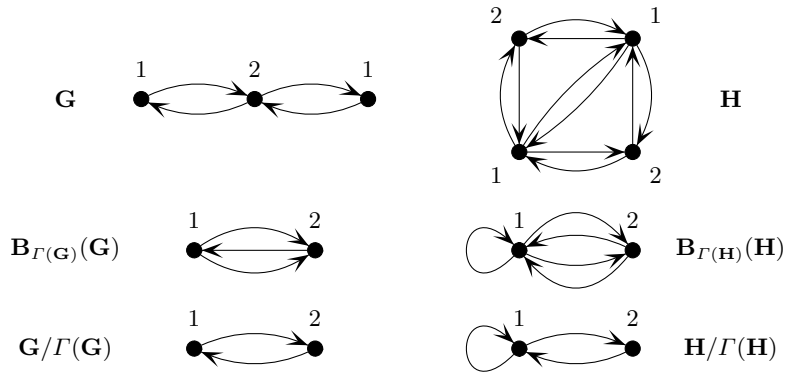

Fig. 1. The digraph $\mathbf{G}$ is fibred over $\mathbf{B}_{\Gamma(\mathbf{G})}(\mathbf{G})$ via the homomorphism $\varphi_{\mathbf{G}}$ that maps each vertex of $\mathbf{G}$ labelled i to the unique vertex labelled i of $\mathbf{B}_{\Gamma(\mathbf{G})}(\mathbf{G})$. The digraph $\mathbf{H}$ is a covering of $\mathbf{B}_{\Gamma(\mathbf{H})}(\mathbf{H})$ via the homomorphism $\varphi_{\mathbf{H}}$ defined in the same way. The digraph $\mathbf{G}/\Gamma(\mathbf{G})$ (resp. $\mathbf{H}/\Gamma(\mathbf{H})$) is the digraph whose vertices and arcs correspond to equivalence classes of vertices and arcs of $\mathbf{G}$ (resp. $\mathbf{H}$) under the action of $\Gamma(\mathbf{G})$ (resp. $\Gamma(\mathbf{H})$).

2.3 Fibrations, Coverings and Automorphisms

We now describe some properties of the relations that exist between fibrations and the automorphisms of a digraph. These results are described and proved in [BV02].

An *automorphism* σ of a digraph $\mathbf{G}$ is an isomorphism from the digraph $\mathbf{G}$ onto itself. Consider a subgroup Γ of the group $\Gamma(\mathbf{G}) = Aut(\mathbf{G})$ of the automorphisms of a digraph $\mathbf{G} = (G, \lambda)$; we will denote by Id the identity automorphism of $\mathbf{G}$. The action of this group on $\mathbf{G}$ induces an equivalence relation over the vertices and the arcs of $\mathbf{G}$: for each $x, x' \in V(G) \cup A(G), x \sim_\Gamma x'$ if there exists $\sigma \in \Gamma$ such that $\sigma(x) = x'$. The equivalence class of x is called the orbit of x and is denoted by $[x]_\Gamma$. Recall that an automorphism of (G, λ) must preserve the labelling, and therefore for all elements $x_1, x_2 \in [x]_\Gamma$, $\lambda(x_1) = \lambda(x_2)$. If $\Gamma = \Gamma(\mathbf{G})$, we will note $x \sim x'$ (resp. $[x]$) for $x \sim_\Gamma x'$ (resp. $[x]_\Gamma$).

Remark 1. For all vertices $v, v' \in V(G)$, if $v \sim_\Gamma v'$, then there is a label-preserving bijection between the incoming arcs of v and the incoming arcs of v'.

We will now describe two kinds of constructions. The first one allows us to build a digraph $\mathbf{B}_\Gamma(\mathbf{G})$ from a digraph $\mathbf{G}$ such that $\mathbf{G}$ is fibred over $\mathbf{B}_\Gamma(\mathbf{G})$. The second one allows us to build the quotient-graph $\mathbf{G}/\Gamma$. Examples are presented in Figures 1 and 2 where $\Gamma = \Gamma(\mathbf{G})$.

From the relation $\sim_\Gamma$, we construct the directed graph $B_\Gamma(\mathbf{G})$ defined as follows: $V(B_\Gamma(\mathbf{G}))$ is the set of the equivalence classes of $V(G)$ under the action of Γ and there are as many arcs from $[v]_\Gamma$ to $[w]_\Gamma$ as each vertex in $[w]_\Gamma$ has predecessors in $[v]_\Gamma$. Due to Remark 1, this does not depend on the choice of the element of $[w]_\Gamma$. We define the labelling ν of $B_\Gamma(\mathbf{G})$ by $\nu([v]_\Gamma) = \lambda(v)$ for each $v \in V(G)$. We label the arcs from $[v]$

to $[w]$ with the labels of the arcs from the elements of $[v]$ to w in $\mathbf{G}$. By Remark 1, there exists a fibration φ from $\mathbf{G}$ to $(B_\Gamma(\mathbf{G}), \nu)$.

We consider also the quotient-graph $\mathbf{G}/\Gamma$ whose vertices and arcs are the equivalence classes of the vertices and the arcs of G under the action of Γ and whose labelling μ is defined by $\mu([x]_\Gamma) = \lambda(x)$ for each $x \in V(G) \cup A(G)$. There exists a natural surjective homomorphism from $(B_\Gamma(\mathbf{G}), \nu)$ to $\mathbf{G}/\Gamma$ which is the identity on the vertices and which maps an arc a to $[a]_\Gamma$ (a can be seen as an arc of G).

We say that a subgroup Γ of $\Gamma(\mathbf{G})$ acts *freely* on $\mathbf{G}$ if for each $x, y \in V(G) \cup A(G)$, there is at most one $\sigma \in \Gamma$ such that $\sigma(x) = y$. Equivalently, Γ acts freely on $\mathbf{G}$ if and only if for each $\sigma \in \Gamma \setminus \{Id\}$, σ has no fixpoint.

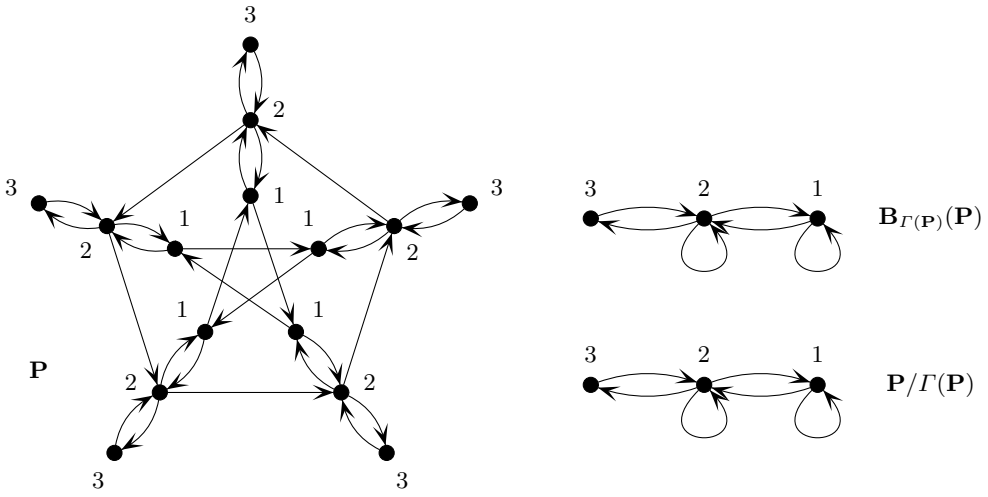

Fig. 2. The labelled digraph $\mathbf{P}$ is a covering of $\mathbf{P}/\Gamma(\mathbf{P})$: the automorphism group of $\mathbf{P}$ contains only well-balanced automorphisms.

In the following, we will use a particular class of automorphisms: the class of *well-balanced* automorphisms. These automorphisms have been introduced by Bougé in [Bou88] to study the importance of the guards in CSP through the symmetric election problem. In [Pal03], Palamidessi uses also well-balanced automorphisms to study the same problem in order to give a hierarchy between different subsets of the π-calculus.

Definition 5. *An automorphism σ of a digraph $\mathbf{G}$ is* well-balanced *if there exists an integer q such that for each vertex or arc x of $\mathbf{G}$, $|\{\sigma^k(x) \mid k \in \mathbb{N}\}| = q$.*

Equivalently, σ is well-balanced if and only if the subgroup Γ_σ generated by σ acts freely on $\mathbf{G}$.

The group Γ contains only well-balanced automorphisms if and only if Γ acts freely on $\mathbf{G}$. Thanks to this equivalence and the results of Boldi and Vigna [BV02], we have the following property.

Proposition 2. *Given any strongly connected digraph $\mathbf{G}$, the quotient projection $\Gamma : \mathbf{G} \to \mathbf{G}/\Gamma$ is a covering projection if and only if for each $\sigma \in \Gamma$, σ is well-balanced.*

3 Impossibility Result

The following proposition gives a necessary condition that the distributed mobile environment $(G, \mathcal{E}, p)$ must satisfy if there exists an election algorithm for $(G, \mathcal{E}, p)$ that successfully elects an agent. This necessary condition is equivalent to the one presented in [BFFS03].

Proposition 3. *Consider a graph G and an initial placement of the agents p. If there exists a non-trivial well-balanced automorphism σ of the digraph $\mathbf{G}' = (Dir(G), \chi_p)$, then it is impossible to elect an agent over the graph G with the initial placement of the agents p.*

Proof. Consider a distributed mobile environment $(G, \mathcal{E}, p)$ such that there exists a well-balanced automorphism σ of $\mathbf{G}' = (Dir(G), \chi_p)$ different from Id. There exists an integer $q \geq 2$ such that $\forall v \in V(G), |\{\sigma^k(v) \mid k \in \mathbb{N}\}| = q$.

Suppose that there exists an election algorithm $\mathcal{A}$ for $(G, \mathcal{E}, p)$. Consider a port-labelling such that each port has a unique color. The automorphism σ induces a permutation of the agents and a permutation of the colors of the agents and of the ports, that will also be denoted by σ. These permutations induce equivalence relations on the agents and on the colors.

Initially, an agent r is on v if and only if $\sigma(r)$ is on $\sigma(v)$, and two equivalent agents are not in the same place. Moreover, the information available to $\sigma(r)$ is the same as the one available to the agent r, up to the permutation σ of the colors. We can exhibit an execution of $\mathcal{A}$ such that these properties remain true all along the execution and such that if a message is written with a color c on a vertex v, then the same message is written on $\sigma(v)$ with the color $\sigma(c)$. Note that two equivalent agents should behave in the same way in this execution and consequently, they should not gather in the same place during the execution. Indeed, if this happens, then the symmetry between the two agents can be broken since one of them can access the whiteboard before the other one.

If an agent r writes a message with its color c on the whiteboard of a vertex v, then for each $k \in [1, q-1]$, the agent $\sigma^k(r)$ writes the same message with its color $\sigma^k(c)$ on $\sigma^k(v)$. Suppose now that an agent r leaves a vertex v through a port colored c_1 to arrive in a vertex w through a port colored c_2. Then for each $k \in [1, q-1]$, the information available to $\sigma^k(r)$ is the same as the one available to r (up to the permutation σ^k of the colors), it leaves the vertex $\sigma^k(v)$ through the port colored $\sigma^k(c_1)$ to arrive at the vertex $\sigma^k(w)$ through the port colored $\sigma^k(c_2)$ and obtains the same information from the whiteboard of $\sigma^k(w)$ as r. Moreover, since σ is well-balanced, we can ensure that two equivalent agents do not gather in a same vertex.

Consequently, if during the execution of $\mathcal{A}$, an agent r takes the label *elected*, then the agent $\sigma(r)$ takes also this label, and therefore $\mathcal{A}$ is not an election algorithm. □

Using known results in the message-passing model [BCG+96,CM05,YK96] and the results of [CGMO06], we can show that in the anonymous setting, i.e., when the agents can understand each other but do not have distinct labels, there exists an election algorithm for an environment $(G, \mathcal{E}, p)$ if and only if the labelled digraph $\mathbf{G}' = (Dir(G), \chi_p)$ is symmetric-covering-minimal. Moreover, from Proposition 2, we know that if the symmetric digraph $\mathbf{G}' = (Dir(G), \chi_p)$ admits a non-trivial well-balanced automorphism σ, then $\mathbf{G}'$ is a symmetric covering of $\mathbf{G}'/\Gamma_\sigma$ that is not isomorphic to $\mathbf{G}'$.

Consequently, an interesting corollary of Proposition 3 is that if the election problem cannot be solved on $(G, \mathcal{E}, p)$ in the qualitative setting, then it cannot be solved on $(G, \mathcal{E}, p)$ in the anonymous setting. On the other hand, we will show in the following that this necessary condition is also sufficient.

Since there exist symmetric digraphs that are not symmetric-covering-minimal and that does not admit any non-trivial well-balanced automorphism, such as the graph G of Figure 3, it means that one can solve the election problem in strictly more environments in the qualitative setting than in the anonymous one.

Example 1. On the Figure 3, we present a digraph $\mathbf{G}$ that is not covering minimal, whereas the underlying digraph G does not admit any automorphism.

Consequently, if we consider the distributed mobile environment corresponding to the digraph (G, χ_p) where $\chi_p(v) = 1$ for each $v \in V(G)$, we know from [BCG+96,CM05,YK96,CGMO06] that there does not exist any election algorithm for this environment in the anonymous setting.

Nevertheless, from the results of the following sections, we can show that one can solve election and rendezvous in this particular distributed mobile environment in the model we consider in this paper.

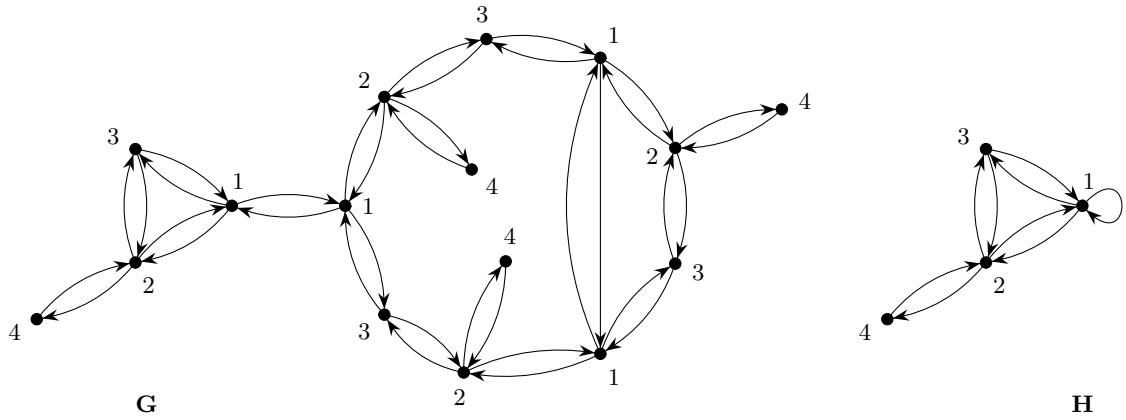

Fig. 3. The digraph **G** is a symmetric covering of **H** via the homomorphism that maps each vertex of **G** labelled i to the unique vertex labelled i of **H**.

4 An Effective Election Algorithm

In this section, we show that the necessary condition described in Proposition 3 is also sufficient. We first show that even if the agents do not understand each other, they can agree on an order on some equivalence classes of vertices and arcs. We then present an algorithm that solves the election problem on any distributed mobile environment $(G, \mathcal{E}, p)$ whenever $(Dir(G), \chi_p)$ does not admit any non-trivial well-balanced automorphism.

4.1 How to order the equivalence classes?

We use the same ideas as Barrière *et al.* [BFFS03] to define a total order between the different equivalent classes. The fact that graphs o size n can be canonically ordered is well-known [McK81]. We construct here a total ordering on digraphs of size n labelled by elements of a totally ordered set.

Consider a labelled digraph $\mathbf{G} = (G, \lambda)$ without multiple arcs where λ is a labelling function from $V(G) \cup A(G)$ to a totally ordered set L with a minimal element $\perp$. We suppose that $\forall x \in V(G) \cup A(G), \lambda(x) \in L \setminus \{\perp\}$.

Let $n = |V(G)|$ and consider an enumeration function num of the vertices (i.e., num is a one-to-one mapping from $V(G)$ onto $[1, n]$). We say that num is an *increasing* enumeration of the vertices if for all vertices $v, v' \in V(G)$, if $num(v) \leq num(v')$, then $\lambda(v) \leq_L \lambda(v')$. Given an increasing enumeration num, we define the adjacency matrix M_{num} as follows: for all vertices v, v', $M_{num}[num(v), num(v')] = \ell$, if there is an arc from v to v' labelled by ℓ, and $M_{num}[num(v), num(v')] = \perp$ otherwise. To this matrix, we associate the word $w(M_{num})$ obtained by the concatenation of the n rows of M_{num}.

To each vertex $v \in V(G)$ (resp. arc $a \in A(G)$), we choose num such that $(num(v), w(M_{num}))$ (resp. $(num(s(a)), num(t(a)), w(M_{num}))$) is minimum for the lexicographic order and associate this value, denoted by $\pi(v)$ (resp. $\pi(a)$), to v (resp. a). Note that there exists an automorphism σ of **G** such that $\sigma(x) = x'$ if and only if $\pi(x) = \pi(x')$. Consequently, this induces a total ordering of the equivalence classes of vertices and arcs: we will write $[x'] \prec [x]$ if $\pi(x)$ is greater than $\pi(x')$ in the lexicographic order.

Remark 2. In the following, we will show that all the agents agree on a total order of the classes and all the agents use the same order. Actually, as it was already explained in [BFFS03], even if the agents cannot agree on an a priori order over the set of colors, they can agree on an order on the different classes, provided that all the agents have the same representation of the graph (up to isomorphism).

In fact, we suppose that each agent has its own totally ordered set isomorphic to $(\mathbb{N}, \leq)$ and each agent can use its own way to compute its order: the algorithm does not make any assumption on the way the order is implemented by each agent.

4.2 An Election Algorithm

In this section, we describe our effective election algorithm. In a first phase all the agents reconstruct the digraph $(Dir(G), \chi_p)$ and check that the election problem can be solved on $(G, \mathcal{E}, p)$. Then, using its knowledge of the graph, each agent constructs the equivalence classes induced by $\Gamma((Dir(G), \chi_p))$. During successive rounds, using the order between the different classes defined above, some agents become passive and get the label *non-elected*, whereas the active agents mark some vertices and some arcs of the digraph to obtain a new labelling μ of the digraph on which all the active agents agree. At the end of the computation, the automorphism group of $(Dir(G), \mu)$ consists only of the identity and each vertex has a unique label. At this point, there is exactly one active agent that is elected. A high level description of the algorithm is presented in Algorithm 1.

Algorithm 1: The Election Algorithm

```
Every agent builds a map of the graph;
Synchronization;
if there exists a non-trivial well-balanced automorphism of G' = (Dir(G), χ_p) then
    Every agent knows that it is impossible to solve the election problem;
else
    Every agent marks as many vertices as possible to construct its initial territory;
    Synchronization;
    /* Initially, all the agents are active and μ = χ_p */
    repeat
        repeat
            The active agents compute the equivalence classes of all the vertices and arcs of G' = (Dir(G), μ);
            if two vertices v, v' (resp. arcs a, a') have the same label μ(v) = μ(v') (resp. μ(a) = μ(a')) and are
            not in the same equivalence class then
                Refine the labelling μ of G' ;
            else if two equivalent active agents do not own the same number of vertices in a given class then
                Refine the labelling μ of G' ;
            else if there exists two equivalent arcs a, a' such that an agent r owns s(a) and t(a) whereas two
            distinct agents r, r' own respectively s(a') and t(a') then
                Refine the labelling μ of G' ;
        until the labelling μ of G' = (Dir(G), μ) cannot be refined ;
        if all the active agents are not in the same class then
            Select active agents;
            The passive agents take the label non-elected;
            The active agents mark the homebases of the passive agents;
            Synchronization;
        else if G' is not a covering of B_{Γ(G')}(G') then
            The active agents mark a class of vertices;
            Synchronization;
        else if G' is not a covering of G'/Γ(G') then
            The active agents mark a class of arcs;
            Synchronization;
        else
            /* In this case, there is exactly one active agent */
            The active agent takes the label elected;
    until An agent is elected ;
```

A Synchronization Procedure. In the algorithm we describe below, we distinguish different rounds. An important point is that an active agent does not enter in a new round if another active agent has not finished the previous one. To be able to avoid this kind of situation, we synchronize the active agents.

Each agent can consistently distinguish its homebase; therefore, we can construct an algorithm such that no agent needs to write anything on its homebase. Moreover, we suppose that each agent has already built its own map of the graph and does not need to write anything on any whiteboard in order to perform a traversal of the graph.

In the following, the active agents will do some traversals of the network and they will store the colors of the marks that appear on each vertex to construct what we will call a *colored map* of the network. The marks that appear in a colored map of an active agent will correspond to marks that have been put by other active agents during the round (but it will not necessary contain all the marks the active agents should put during this round).

In any round of the algorithm described below, the following properties will always be satisfied.

(P1) Each active agent can know from a colored map if any other active agent has marked all the vertices it should have marked during the round.
(P2) In each round of the algorithm, each agent will mark at least one vertex (which is not its homebase).

In the synchronization procedure described below, some active agents will have to wait on some particular vertices for other agents to put (resp. remove) some marks. Each time an agent arrives on a place where it has to wait for a mark to be put (resp. removed), it can immediately continue to execute the procedure if this mark is present (resp. not present).

To synchronize the agents, we proceed as follows. Each round is divided in nine steps and during each round, each active agent r executes the following steps. During the first step, the active agents mark some vertices in order to communicate according to the rules of the algorithm we described below; the eight remaining steps are used to synchronize the active agents.

(Step 1) The agent r marks some vertices (but not its homebase) according to the computation rules of the round. In this step, the property (P2) is always satisfied.
(Step 2) The agent r does a traversal of the network and stores all the colors of the marks that appear on each vertex to construct a colored map of the network.
(Step 3) If there exists another active agent r' that has not finished (Step 1) (the agent r can detect it from the colored map it has of the network, since properties (P1) and (P2) are always satisfied), then the agent r goes to the homebase of r' and waits until the agent r' puts a mark on its homebase. Then the agent r does a traversal of the network and stores all the colors of the marks that appear on each vertex in order to update its colored map of the network.
(Step 4) The agent r puts a mark on its homebase.
(Step 5) The agent r does a traversal of the network and each time it arrives on the homebase of another active agent r', it waits until the agent r' marks its homebase.
(Step 6) The agent r does a traversal of the network and it removes the marks it puts during (Step 1), but not the mark on its homebase.
(Step 7) The agent r does a traversal of the network. Each time it arrives on a vertex that has been marked by another active agent r' during this round (but that is not the homebase of r'), it waits until the agent r' removes its mark.
(Step 8) The agent r removes the mark it puts on its homebase.
(Step 9) The agent r does a traversal of the network. Each time it arrives on the homebase of an active agent r', it waits until the agent r' removes its mark on its homebase.

We can note that the synchronization procedure enables also to erase all the marks that have been put on the vertices during the round, i.e., when one agent has finished Step (9) of a round, then all the marks that have been left by the active agents during this round have been erased. The following proposition ensures that the procedure is indeed a synchronization procedure.

Proposition 4. *Each time an agent starts executing Step* (1) *of the* $i+1$*th round, then each active agent knows what vertices have been marked by the other active agents during Step* (1) *of the* i*th round and all the marks that have been put during the* i*th round have been removed. Moreover, the synchronization procedure avoids any deadlock.*

Proof. First, we show that if an agent is executing some particular step of a round, then one can know what are the possible steps the other agents can perform at this moment.

If an agent is currently performing Steps (1), (2) or (3) (resp. Steps (4) or (5), Steps (6) or (7), Steps (8) or (9)) of a round i, then it means that it has finished Step (9) of the $i-1$th round (resp. Step (3) of the ith round, Step (5) of the ith round, Step (7) of the ith round). Consequently, all the other agents have finished Step (8) of the $i-1$th round (resp. Step (1) of the ith round, Step (4) of the ith round, Step (6) of the ith round).

If an agent has neither finished Step (9) of the $i-1$th round nor Step (1) of the ith round (resp. Steps (2), (3) nor (4) of the ith round, Steps (5) nor (6) of the ith round, Steps (7) nor (8) of the ith round), then it means that no agent can have finished Step (3) of the ith round (resp. Step (5) of the ith round, Step (7) of the ith round, Step (9) of the ith round).

All these properties are summarized in the following table.

If an agent r is performing Step	then any active agent r' has finished Step	and no active agent r' has finished Step
(1) in the ith round,	(8) of the $i-1$th round,	(3) of the ith round.
(2) in the ith round,	(8) of the $i-1$th round,	(5) of the ith round.
(3) in the ith round,	(8) of the $i-1$th round,	(5) of the ith round.
(4) in the ith round,	(1) of the ith round,	(5) of the ith round.
(5) in the ith round,	(1) of the ith round,	(7) of the ith round.
(6) in the ith round,	(4) of the ith round,	(7) of the ith round.
(7) in the ith round,	(4) of the ith round,	(9) of the ith round.
(8) in the ith round,	(6) of the ith round,	(9) of the ith round.
(9) in the ith round,	(6) of the ith round,	(3) of the $i+1$th round.

Suppose that there exists a deadlock in the application of the synchronization procedure, i.e., there exists a moment in the execution where all the agents are blocked at some step of the procedure. In each round, the only steps where an agent can be blocked are Steps (3), (5), (7) and (9).

Suppose that an active agent r is blocked at Step (3) of a round i. Then, we know that all the other active agents have finished Step (8) of the $i-1$th round and consequently, none of them can be blocked at Step (9) of the $i-1$th round and all of them have finished Step (1) of the ith round. Suppose that during the ith round, there exists k_i distinct active agents and let call r_1 the first agent that have finished Step (1) of the ith round, r_2 the second one, r_3 the third one, etc...

We consider the agent r_j and we suppose that for any $j < l \leq k_i$, the agent r_l has performed Step (4) of the ith round (this hypothesis is trivially true for the agent r_{k_i}). Suppose that the agent r_j is waiting at the homebase of another agent r_l during Step (3) of the ith round. If $l < j$, then the agent r_l has finished Step 1 of the ith round before r_j. Consequently, from the colored map r_j constructs during Step (2) of the ith round, r_j knows that r_l has finished Step (1) of the ith round and then r_j cannot wait at the homebase of the agent r_l. Suppose now that $l > j$. We already know that the agent r_l has performed Step (4) of the ith round and then r_l has marked its homebase. Consequently, the mark r_j was waiting for has been put and then r_j cannot be waiting on the homebase of r_l. Then r_j cannot be blocked at Step (3) of the ith round and it has also finished Step (4) of the ith round, since it cannot be blocked at this step.

By induction, it is then easy to prove that no active agent can be waiting for another agent while it is performing Step 3 of a round i.

Suppose now that an active agent r is blocked at Step (5) of a round i. Then all the active agents have finished Step (1) of the ith round and we already know that none of them can be blocked at Step (3) of the ith round. Consequently, each active agent has performed Step (4) of the ith round and then no agent can be waiting for another agent while it is performing Step (5) of the ith round.

Suppose now that an active agent r is blocked at Step (7) (resp. Step (9)) of a round i. Then all the active agents have finished Step (4) (resp. Step (6)) of the ith round and consequently, no active agent can be blocked at step (5) (resp. (7)) of the ith round. Consequently, each active agent has finished Step (6) (resp. Step (8)) of the ith round and then no agent can be waiting for another agent while it is performing Step (7) (resp. Step (9)) of the ith round.

Consequently, there is no deadlock in the application of the synchronization procedure. Moreover, each time an agent starts the computation of a round $i+1$, i.e., each time it performs Step (1) of a round $i+1$, all the active agents have finished Step (8) of the ith round and consequently all the marks that have been put during the ith round have been removed. Furthermore, since all the active agents have have finished Step (3) of the ith round, they get all the information they should gather during the execution of the ith round. □

Initialization. During the first phase of the algorithm, each agent reconstructs the graph with the position and the colors of the different homebases. Using the whiteboards, each agent performs a depth first traversal of the graph.

Since each agent can distinguish all the homebases, we suppose that during this traversal, the agents do not write anything on the whiteboard of any of the homebases. Once an agent has reconstructed the whole graph, it performs a traversal of the network using the information it has stored to erase what it has written on the whiteboards. At this point, each agent puts a mark on the homebase of another agent (note that a homebase may be marked by two or more agents).

During this first phase, no agent has written anything on its homebase. Furthermore, an agent has finished performing this phase if and only if it has marked the homebase of another agent and this can be checked from a colored map of the network. Moreover, at the end of this phase, each agent has reconstructed a map of the network and it knows the position of all the homebases. We can therefore use the synchronization procedure defined above at this point.

If the digraph $(Dir(G), \chi_p)$ admits a well-balanced automorphism σ different from Id, then each agent detects it and declares that the election problem is unsolvable in this environment. We will now suppose that $(Dir(G), \chi_p)$ does not admit such an automorphism.

Once the graph is known by all the agents, each agent tries to mark as many vertices of the networks as possible. It does a traversal of the network and each time it arrives on a vertex that is not a homebase, it performs one of the two following actions. Either the whiteboard is blank and it puts a mark with its color on the whiteboard, or there is already a mark on the whiteboard and it stores the color of the mark. Once an agent has finished this traversal, it puts a mark on the homebase of another agent. Again we use the synchronization procedure at this point. Then each agent is aware of the different vertices marked by the other agents during this round.

At the end of this phase, each agent reconstructs a graph where all the vertices are colored (they belong to the agent that has this color) and it knows the position and the color of the homebases of all the other agents. The *territory* of an agent is the set of all places this agent owns.

How can the agents increase their territory? During the different phases of the algorithm, some agents become passive whereas the others continue to execute the protocol in order to elect one of them. In our algorithm, in order to break the symmetry between the agents, all the vertices must belong to one active agent, and all the active agents must agree on which agent a vertex belongs to. During the initialization, each vertex is marked by one agent and we say that it belongs to this agent. Once an agent becomes passive, the vertices that belonged to this agent must be given to another agent.

Once a selection between agents is done, the agents that become passive take the label *non-elected* and remain passive until the end of the algorithm, whereas the others try to mark the homebases of these agents that have just become passive. Each active agent knows what are the colors of the other active agents. From its representation of the graph, each active agent can reach the homebases of the passive agents.

The first agent that reaches such a homebase during this round puts a mark with its color on the vertex. The other agents (there is already a mark on the homebase when they reach it) store the color of the agent that owns this vertex (i.e., the color of the mark). Again, at the end of its traversal of the graph, each

agent puts a mark on the homebase of another active agent. Therefore, each active agent can detect from a colored map if another active agent has finished this phase. Then the active agents apply the synchronization procedure.

If an agent has marked the homebase of a passive agent, then all the vertices that were belonging to this passive agent belong now to this active agent. In this way, all the active agents agree on the color of the agent that owns any vertex of the graph.

How to refine the labelling μ? During the execution of the algorithm, the agents mark vertices and arcs to break the symmetry that may exist in the network. In this way, at each round, numbers will be associated to some vertices and arcs and we will obtain a labelling of the graph μ. Initially, all the homebases have the label 1 whereas all the other vertices have the label 0 and all the arcs are labelled 0.

At the beginning of each round, from its representation $(Dir(G), \mu)$ of the graph, each agent computes the value $\pi(v)$ (resp. $\pi(a)$) for each vertex $v \in V(Dir(G))$ (resp. for each arc $a \in A(Dir(G))$). We say that two agents are equivalent if their homebases are in the same equivalence class, and we use the order $\prec$ on the homebases of the agents to order the classes of agents.

Since all the agents agree on the order to compare the equivalence classes, we can use the following procedure. If there exist two vertices v, v' such that $\mu(v) = \mu(v')$ and $\pi(v) \neq \pi(v')$, then let m be the lowest number such that there exist v, v' with $\mu(v) = \mu(v') = m$ and $[v] \prec [v']$. Suppose that there exist exactly j classes $\{[v_i] | i \in [1, j]\}$ such that $\mu(v_i) = m$ and $[v_1] \prec [v_2] \prec \cdots \prec [v_j]$. For each vertex $v \in [v_i]$ with $i < j$, we define $\mu'(v) = q + i$, where q is the greatest label that appears on a vertex in (G, μ). The labels of the other vertices are not changed.

We apply the same method to arcs using the order we have on the classes of arcs, i.e., the lexicographic order over the $\pi(a)$. Thanks to this procedure, two arcs that are not in the same class are given distinct labels.

We repeat this procedure, until all the vertices (resp. all the arcs) that have the same label are in the same equivalence class.

If some active agents do not own the same number of vertices in a given class. We consider now a configuration such that two vertices (resp. arcs) in different classes have different numbers. Consider a class of agents $[r]$ and a class of vertices $[v]$. We define $NotBalanced([r], [v])$ to be false if all the agents of $[r]$ own the same number of vertices in $[v]$, and true otherwise. If there exist $[r], [v]$ such that $NotBalanced([r], [v])$ is true, then we apply the following technique to split some class of vertices.

Consider the minimum class $[r]$ of agents, according to $\prec$, such that there exists a class $[v]$ of vertices satisfying $NotBalanced([r], [v])$. Consider the minimum class of vertices $[v]$ such that $NotBalanced([r], [v])$ is true. In this case, we give different numbers to the homebases of the agents that do not own the same number of vertices in $[v]$. We subdivide the class $[r]$ into a partition $R_1, \ldots, R_j$ such that the agents in R_i own strictly more vertices in $[v]$ than the agents in $R_{i'}$ when $i < i'$. Using the same technique as before, we give different numbers to the homebases of the agents that are not in the same R_i and then obtain a new representation of the digraph $(Dir(G), \mu')$.

Then, the agents try to refine again this new labelling.

How to split the arc classes thanks to the colors of their ends? We will say that an arc a *belongs* to an agent r, if r owns $s(a)$ and $t(a)$. Otherwise, the arc is such that $s(a)$ belongs to an agent r_1 and $t(a)$ to a distinct agent r_2. We will say that this arc is *shared* by r_1 and r_2. If there exists a class of arcs $[a]$ such that some arcs of $[a]$ belong to some agents, whereas the other arcs are shared by distinct agents, then we apply the following technique.

Consider a class of arcs a such that for each class $[a'] \prec [a]$, either $[a']$ contains only arcs that belong to some agents or $[a']$ contains only arcs shared by different agents. We suppose also that $[a]$ contains arcs that belong to some agents and arcs that are shared. All the arcs in $[a]$ that are shared by distinct agents are relabelled $q + 1$, where q is the greatest label that appears on an arc in $(Dir(G), \mu)$.

Then, the agents try to refine again this new labelling.

Remark 3. Note that to refine the labelling μ of the graph $(Dir(G), \mu)$, the agents do not have to move in the network. All these computations can be performed only with the information available to each agent. Since all the agents have the same colored map of the network (up to isomorphism) and execute the same algorithm, we know that all the agents will compute exactly the same refined labelling.

If some active agents are in different classes. We suppose now that the labelling μ of $(Dir(G), \mu)$ cannot be refined any more, i.e., two vertices (or arcs) have the same number if and only if they are in the same equivalence class, all the active agents own the same number of vertices in each class and two arcs in a same class are either both owned by some agents, or both shared by distinct agents.

At this point, if the active agents are not in the same equivalence class, we are able to select some agents. Consider all the equivalence classes of active agents that contain a minimal number of agents. Among these classes, we select the class $[r]$ such that $\pi(v)$ is minimal, where v is the homebase of r. The agents that do not belong to this class take the label *non-elected* and become passive. The agents of the class $[r]$ remain active and try to increase their territory as explained above. Then they try to refine again the labelling μ.

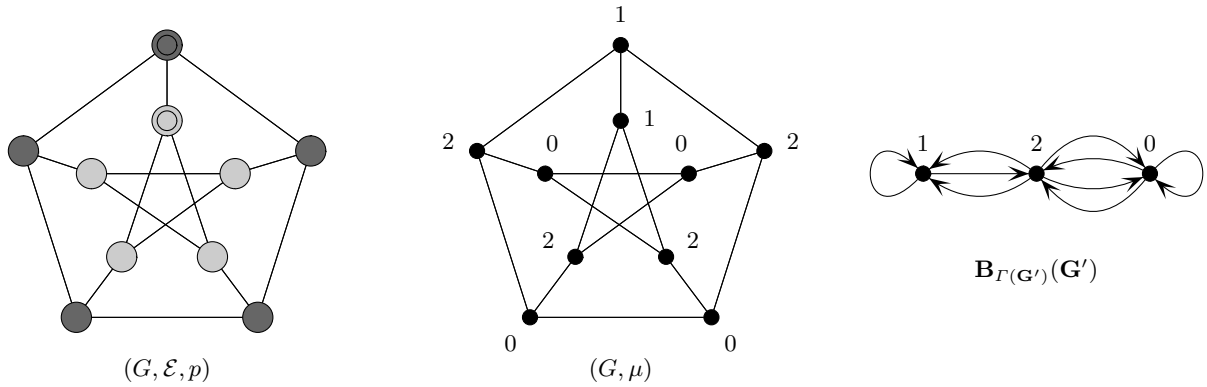

Fig. 4. A colored map of a distributed mobile environment $(G, \mathcal{E}, p)$ with two agents, the corresponding graph (G, μ) and the digraph $\mathbf{B}_{\Gamma(\mathbf{G}')}(\mathbf{G}')$ where $\mathbf{G}' = (Dir(G), \mu)$. In the representation of $(G, \mathcal{E}, p)$ the homebases are represented by double-circled vertices and each vertex belongs to the agent whose homebase has the same color. For sake of clarity, the labels of the arcs in (G, μ) and $\mathbf{B}_{\Gamma(\mathbf{G}')}(\mathbf{G}')$ do not appear.

If $\mathbf{G}' = (\mathbf{Dir(G)}, \boldsymbol{\mu})$ is not a covering of $\mathbf{B}_{\boldsymbol{\Gamma(\mathbf{G}')}}(\mathbf{G}')$. There exist some configurations where it is impossible to select some agents just by using the representation the agents have of the graph, because there is too much symmetry in the graph.

Example 2. Consider the distributed mobile environment corresponding to the graph represented on Figure 4. We suppose that the agents agree on the colored map of the network and on the graph $(Dir(G), \mu)$. We can observe that the labelling μ cannot be refined and that $(Dir(G), \mu)$ does not admit any well-balanced automorphism.

We now explain how active agents can break these symmetries by marking some vertices or arcs. We suppose that the labelling μ cannot be refined any more and that all the active agents belong to the same equivalence class.

All the active agents agree on the graph $\mathbf{G}' = (Dir(G), \mu)$. All these agents consider the automorphism group $\Gamma(\mathbf{G}')$ and construct the graphs $\mathbf{B}_{\Gamma(\mathbf{G}')}(\mathbf{G}')$ and $\mathbf{G}'/\Gamma(\mathbf{G}')$. We already know that $\mathbf{G}'$ is fibred over $\mathbf{B}_{\Gamma(\mathbf{G}')}(\mathbf{G}')$. If $\mathbf{G}'$ is not a covering of $\mathbf{B}_{\Gamma(\mathbf{G}')}(\mathbf{G}')$, it implies that there exist two classes of vertices $[v]$ and $[v']$ such that $|[v]| \neq |[v']|$. Let $[r]$ be the class of the homebases of the active agents.

Consider a class $[v]$ such that for each class $[v'] \prec [v]$, $|[v']| = |[r]|$ and $|[v]| \neq |[r]|$. We already know that each active agent owns the same number of vertices in $[v]$ and therefore $|[r]|$ divides $|[v]|$. Each active agent then marks a vertex it owns that is in $[v]$. An agent has finished this round if and only if it has marked exactly one vertex in $[v]$: it can be detected from a colored map of the graph. Then, the agents synchronize.

At the end of this round, all the agents give the number $q+1$ to the vertices that have just been marked, where q is the greatest label that appears on a vertex in $(Dir(G), \mu)$.

Using this new labelling μ', the active agents try to refine the labelling μ', as explained above.

If $\mathbf{G'} = (Dir(G), \mu)$ is not a covering of $\mathbf{G'}/\Gamma(\mathbf{G'})$. We suppose now that the labelling μ cannot be refined any more and that all the active agents belong to the same equivalence class. We suppose also that $\mathbf{G'}$ is a covering of $\mathbf{B}_{\Gamma(\mathbf{G'})}(\mathbf{G'})$ but not of $\mathbf{G'}/\Gamma(\mathbf{G'})$. It means that all the equivalence classes of vertices have the same size s, but there exists an equivalence class of arcs $[a]$ such that $|[a]| > s$. Instead of marking vertices, we mark arcs in this round.

Each class $[a]$ of arcs of $\mathbf{G'}$ corresponds to exactly one arc in $\mathbf{G'}/\Gamma(\mathbf{G'})$. Consider the class of arcs $[a]$ such that for each class $[a'] \prec [a]$, $|[a']| = s$ but $|[a]| > s$. We already know that each active agent owns exactly one vertex in $[s(a)]$ and one vertex in $[t(a)]$. Since $|[a]| > s$ and since two arcs in the same class are either both owned by an agent or both shared by distinct agents, we know that each arc in $[a]$ is shared.

To select arcs from $[a]$, each agent r just chooses one arc a_r in $[a]$ such that $s(a_r)$ belongs to r and then puts a mark with its color on $t(a_r)$. An agent has finished this round if and only if it has marked exactly one vertex: it can be detected from a colored map of the graph. Then, the agents synchronize. Once an agent knows what vertices have been marked by the other agents, it knows what are the arcs that have been marked.

At the end of this round, all the agents give the number $q+1$ to the arcs that have just been marked, where q is the greatest label that appears on an arc in $(Dir(G), \mu)$.

Using this new labelling μ', the active agents try to refine again the labelling μ', as explained above.

If $\mathbf{G'}$ is a covering of $\mathbf{G'}/\Gamma(\mathbf{G'})$. At this point, $\mathbf{G'} = (Dir(G), \mu)$ is a covering of $\mathbf{G'}/\Gamma(\mathbf{G'})$. From Proposition 2, it implies that $\Gamma(\mathbf{G'})$ contains only well-balanced automorphisms, and since we already know that there is no well-balanced isomorphism of $(Dir(G), \chi_p)$ different from Id, we have $\Gamma(\mathbf{G'}) = \{Id\}$. Consequently, there is exactly one active agent, since the set of active agents is an equivalence class of the relation induced by $\Gamma(\mathbf{G'})$ and this agent takes the label *elected*.

4.3 The Characterization

In Section 3, we have shown that if the graph $(Dir(G), \chi_p)$ admits a well-balanced automorphism, then it is impossible to solve the election problem on $(G, \mathcal{E}, p)$. The algorithm described in Section 4.2 is an algorithm that answers that it is impossible to solve the problem if the graph $(Dir(G), \chi_p)$ admits a well-balanced automorphism, and otherwise it successfully elects an agent: it is an effective algorithm. We have therefore proved the following theorems.

Theorem 1. *There exists an election algorithm for a distributed mobile environment $(G, \mathcal{E}, p)$ if and only if $(Dir(G), \chi_p)$ does not admit a non-trivial well-balanced automorphism.*

Proof. The necessary condition comes from Proposition 3.

For the sufficient condition, we just have to prove that Algorithm 1 elects a leader whenever the necessary condition is satisfied. Consider a distributed colored mobile environment $(G, \delta, \mathcal{E}, p, color)$ such that $(Dir(G), \chi_p)$ does not admit a non-trivial well-balanced automorphism. Consider an execution of Algorithm 1 on $(G, \delta, \mathcal{E}, p, color)$. From Proposition 4, we know that no deadlock can occur within a round.

Initially, all the agents agree on the labelling $\mu = \chi_p$ of $\mathbf{G'}$. After the initialization, each vertex belongs to exactly one agent and each agent knows the owner of each vertex of the graph.

From Remark 3, we know that each time the active agents refine the labelling μ of $\mathbf{G'}$, they agree on the new labelling μ' of $\mathbf{G'}$ provided that they agreed on the labelling μ. Since the territory of any agent has not

changed, the active agents also know the territory of any other active agent and each vertex still belongs to exactly one active agent.

Furthermore, each time the active agents mark some vertices and arcs, they also agree on the new labelling μ' of $\mathbf{G}'$ whenever they agreed on the labelling μ of $\mathbf{G}'$. Since the territory of any agent has not changed, the active agents also know the territory of any other active agent and each vertex still belongs to exactly one active agent.

Finally, when a selection is made between active agents, the labelling μ is not modified and consequently, we know that the active agents always agree on the labelling μ of $\mathbf{G}'$. It is easy to see that after such a phase, once the active agents have marked the homebases of agents that have just been defeated, then each vertex belongs to exactly one active agent and each active agent knows the territory of any other active agent.

Consequently, by induction, we know that after each round, all the active agents agree on the labelling μ of $\mathbf{G}'$, each vertex belongs to exactly one active agent and each active agent knows the territory of any other active agent.

Note that each time the labelling μ of $\mathbf{G}'$ is refined, then the number of labels given to vertices and arcs by active agents increases. Since the graph G is finite, we know that the labelling refinment procedure will always terminate. Moreover, each time a selection is made between active agents, the number of active agents decrease. Furthermore, each time active agents mark vertices or arcs, the number of labels given to vertices and arcs by active agents increases. Consequently, there won't be any infinite execution of the algorithm.

Consider now the final configuration of the environment reached after the execution. Suppose that some agent is in the state *elected*. It means that $\mathbf{G}'$ is a covering of $\mathbf{G}'/\Gamma(\mathbf{G}')$ and since $(Dir(G), \chi_p)$ does not admit any non-trivial well balanced automorphism, then from Proposition 2, $\Gamma(\mathbf{G}') = \{Id\}$. Consequently, there is exactly one active agent that took the label *elected*, i.e., the execution has successfully elected an agent.

Suppose now that in the final configuration, no agent has took the label elected. Since $(Dir(G), \chi_p)$ does not admit any non-trivial well balanced automorphism, we know that all the active agents are executing the loop **repeat** ... **until** *An agent is elected*. Since the execution has terminated, then the labelling μ of $\mathbf{G}$ cannot be refined any more and we know that all the agents are in the same equivalence class. In this case, we know that $\mathbf{G}'$ is a covering of $\mathbf{B}_{\Gamma(\mathbf{G}')}(\mathbf{G}')$, since otherwise the active agents can select vertices. We also know that $\mathbf{G}'$ is a covering of $\mathbf{G}'/\Gamma(\mathbf{G}')$, since otherwise the active agents can select arcs. Consequently, there is exactly one active agent and this active agent took the label *elected*, which leads to a contradiction.

Consequently, any execution of Algorithm 1 on $(G, \delta, \mathcal{E}, p, color)$ succesfully elects an agent. Then Algorithm 1 is an election algorithm for $(G, \delta, \mathcal{E}, p, color)$.

The proof of the following theorem follows from Theorem 1 and the fact that the agents can easily reconstruct a map of the graph, as explained in the description of the algorithm.

Theorem 2. *Algorithm 1 is an effective election algorithm.*

4.4 Complexity of the algorithm

The main goal of this work was to design an effective election algorithm in our model, but it can also be interesting to have a look on the complexity of the algorithm. The traditional coast measures for mobile agents are the number of agents moves (edge traversals) and the amount of time.

To determine the amount of time, we make the following assumptions : the time needed by an agent to perform computations is equal to zero time unit, whereas the time needed by an agent to traverse an edge is equal to one time unit. The time complexity of the algorithm is the time consumed by an execution of the algorithm under these assumptions. In other words, we do not take into account the computation time of each agent, and we suppose that all the agents traverse all the edges at the same speed. Note that the correctness of our algorithm does not rely on these assumptions.

The following proposition gives a bound on the worst case complexity of our algorithm.

Proposition 5. *Consider a distributed mobile environment $(G, \mathcal{E}, p)$ where $|V(G)| = n$, $|E(G)| = m$ and $|\mathcal{E}| = k$. If the election problem cannot be solved in this environment, the agents detect it in $O(mk)$ moves in time $O(m)$. Otherwise, the protocol successfully elects a leader with $O(mn \log k)$ moves in time $O(mn)$.*

Proof. To perform the first traversal of the graph, each agent will traverse each edge exactly twice: it can do it in $O(m)$. After this traversal, each agent has a map of the graph, and it can check if the problem can be solved. If it is not possible to solve the problem, the agents just need to do this traversal of the graph to give a correct output: it can be done in $O(mk)$ moves in time $O(m)$.

In the following, each time an agent has to check the state of the different vertices, it can use the map of the graph it has reconstructed: it can do it in $O(n)$ moves. The initialization phase of the algorithm can be performed in $O(mk)$ moves in time $O(m)$.

We will consider different phases in the execution of the algorithm. Each time we select a class of agents, we go from phase i to phase $i+1$. Let k_i denotes the number of active agents at phase i: we know that $k_{i+1} \leq k_i/2$, and therefore, there is at most $\log k$ phases during each execution of the algorithm.

To perform the synchronization between agents, each active agent has to do a few traversals of the graphs that can be performed in $O(n)$, and therefore, each time the synchronization procedure is applied during the phase i, it can be done in $O(nk_i)$ moves with time $O(n)$. The time needed by the agents to mark the homebases of the agents that become passive at the end of the phase $i-1$ and to synchronize can be done in $O(nk_i)$ moves with time $O(n)$. The total time complexity for the territory acquisition is $O(n \log k)$.

During the whole execution, the agents will perform at most n times the procedure to mark vertices and m times the procedure to mark arcs. Each time the active agents mark some vertices or some arcs, each agent do its traversal and apply the synchronization procedure in $O(n)$ time (note that even when the agents mark arcs, they do a traversal to visit all the vertices and not all the arcs). The time complexity is therefore $O(nn) + O(mn) + O(n \log k) = O(mn)$.

During each phase, some vertices and arcs can be marked. Consider a class $[x]$ of vertices or arcs of size s at the beginning of the phase i. If during the round, there exists a class of vertices or arcs of size lower than k_i, some agents will be selected and therefore we go to phase $i+1$. Consequently, during the phase i, each active agent mark at most s/k_i vertices or arcs in $[x]$, They can do it in $O(s/k_i * n * k_i) = O(sn)$ moves. Since the sum of the sizes of all classes of vertices and edges is $O(m)$, the agents need $O(mn)$ moves in each phase to mark vertices and arcs and to synchronize.

Consequently, the total number of moves in one phase is $O(nk_i) + O(nn) + O(mn) = O(mn)$. Therefore, the total number of moves to execute the algorithm is $O(mn \log k)$. □

5 Final Remarks

In this paper, we have characterized the distributed mobile environments where the election problem can be solved. The algorithm we present to solve the problem is an effective algorithm: we give a positive answer to the question stated in [BFFS03]. Since election and rendezvous are two equivalent problems, we also have characterized the mobile distributed environments where the rendezvous problem can be solved in our model.

One can observe that the algorithm and the results presented in this paper can be adapted to other kinds of distributed mobile agent systems. Indeed, the algorithm rely on the two following assumptions :

- each agent can reconstruct a map of the graph with the position of the homebases and can know the exact position of its homebase,
- it is possible to identify the author of a message written on the whiteboard of some vertex.

Consequently, if we consider an anonymous mobile agent system where each agent knows initially the network (a map of the graph with the port-numbering function) and its position in the network, then we can solve election and rendezvous if and only if the digraph $(Dir(G), \chi_p)$ does not admit any non-trivial well-balanced automorphism. In order to identify the author r of a message written at a vertex v, it is sufficient that the message contains the port-numbers that appear on a path between the vertex v and the homebase $p(r)$ of r. Therefore, if we consider anonymous systems with sense of direction, as considered by Barrière *et al.* in [BFFS06], we have the same results as in our model, since sense of direction enables each agent to build a map of the network.

In our model, the labels are mutually incomparable. In the quantitative world, we usually suppose that there is always an order between two elements. Between these two extremal cases (a total order or no order

at all), one can wonder whether the existence of a partial order between labels really modifies the conditions presented here.

We can also consider a network where the labels cannot be compared, but where we do not assume that each label is unique: is there a characterization of the graphs where the election problem can be solved if there are different agents with the same label? Note that if all the agents have the same label, we are in the anonymous setting, and in this case, we already know that there is no effective algorithm for the class of all graphs. We can still wonder whether there is a characterization of classes of graphs that admit an universal election algorithm or an effective election algorithm.

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.

[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.

[BFFS03] L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Can we elect if we cannot compare? In *Proc. of the 15th annual ACM Symposium on Parallel Algorithms and Architectures, (SPAA 2003)*, pages 324–332. ACM Press, 2003.

[BFFS06] L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Rendezvous and election of mobile agents: impact of sense of direction. *Theory of Computing Systems*, to appear, 2006.

[Bou88] L. Bougé. On the existence of symmetric algorithms to find leaders in networks of communicating sequential processes. *Acta Informatica*, 25(2):179–201, 1988.

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

[CGMO06] J. Chalopin, E. Godard, Y. Métivier, and R. Ossamy. Mobile agent algorithms versus message passing algorithms. In *Proc. of the 10th International Conference on Principles of Distributed Systems (OPODIS 2006)*, volume 4305 of *Lecture Notes in Computer Science*, pages 187–201. Springer-Verlag, 2006.

[Cha06] J. Chalopin. Election in the qualitative world. In *Proc. of Structural Information and Communication Complexity, 13th International Colloquium (SIROCCO 2006)*, volume 4056 of *Lecture Notes in Computer Science*, pages 85–99. Springer-Verlag, 2006.

[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.

[DFNS05] S. Das, P. Flocchini, A. Nayak, and N. Santoro. Distributed exploration of an unknown graph. In *Proc. of Structural Information and Communication Complexity, 12th International Colloquium (SIROCCO 2005)*, volume 3499 of *Lecture Notes in Computer Science*, pages 99–114. Springer, 2005.

[DFP03] A. Dessmark, P. Fraigniaud, and A. Pelc. Deterministic rendezvous in graphs. In *Proc. of the 11th Annual European Symposium (ESA 2003)*, volume 2832 of *Lecture Notes in Computer Science*, pages 184–195, 2003.

[KKR06] E. Kranakis, D. Krizanc, and S. Rajsbaum. Mobile agent rendezvous: A survey. In *Proc. of Structural Information and Communication Complexity, 13th International Colloquium (SIROCCO 2006)*, volume 4056 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 2006.

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

[McK81] B. D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

[Pal03] C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π-calculus. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.

[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.