What Do We Need to Know to Elect in Networks with Unknown Participants?

Jérémie Chalopin, Emmanuel Godard and Antoine Naudin*

LIF, Université Aix-Marseille and CNRS

Abstract. A network with unknown participants is a communication network where the processes have very partial knowledge of the system. Nodes do not know the full set of participating nodes and some nodes do not even know the full set of nodes they can communicate directly with. It is a "contact list" like network where the initial communication is possibly asymmetric and one can communicate with an unknown neighbour only if one has been first contacted by this neighbour. This model is quite natural and of important theoretical interest. It has also proved useful for the study of bootstrapping mobile ad hoc networks. In this paper, we investigate the classical Leader Election problem in general networks with unknown participants.

We give the first necessary and sufficient condition on global knowledge that nodes should be provided in order to solve Election problem. Since Election problem is a useful benchmark in distributed computability investigations, this result could lead to a complete characterisation of what is solvable in networks with unknown participants.

Keywords: Distributed Algorithm, Message Passing, Leader Election, Distributed Computability, Unknown Participants, Structural Knowledge

1 Introduction

A Natural Model for Distributed Computations. Distributed systems are pervasive and recently more and more interest has been in studying systems that range from dynamic to highly dynamic. Surprisingly there have been few studies of some models that are static but where the local connectivity evolves in a light way during the computation. The following "contact list model" is a fairly natural model related to the communication namespace necessitated by distributed computing.

Consider a set of participants that communicate with phones. Initially, everybody knows a subset of the phone numbers of other people via its personal contact list. It could even be not symmetric. Namely Alice could have the phone number of Bob, whereas Bob would not know the one of Alice. In this situation, Bob cannot call Alice, he does not even know that Alice is participating. Only when Alice has first contacted Bob (and the phone number of Alice is registered

* Work partially supported by Macaron project (ANR JCJC 13-JS02-0002-01)

by the phone of Bob), then Bob can possibly call Alice. In this setting, the contact list (that is the set of neighbours) of Bob increases during the computation. This is a fairly natural model that exhibit general and interesting properties : connectivity is directed, adjacency is initially limited but increases over time, this increase is not automatic and depends of the communications that take place, everybody calling at its own pace. That is the system is asynchronous.

This model is natural and realistic, it is a slight variation of the model introduced in [CSS04] to investigate the self-organized bootstrapping of mobile ad hoc networks (MANETs). Conjointly with the fact that this model has not been very much studied, such systems exhibit new interesting properties related to the theory of distributed computability. We describe them more precisely later.

The Formal Model. The underlying communication graph is an arbitrary undirected graph denoted by G. Nodes are endowed with *identities*, they communicate by *messages*. Their neighbours are addressed with *port numbers* but this port numbering is not explicitly available to the nodes. Initially, a node can send messages only to a (*possibly strict*) *subset* of its actual neighbours in G: its *contact list* of neighbours, this defines the initial *directed* graph G^0. This contact list of neighbours will be extended whenever a message is received from an "unknown" in-neighbour. Therefore, at the end of the computation, the possible communication graph corresponds to G, as G is the undirected version of the initial digraph G^0. Note that the network is reliable but asynchronous: messages are always delivered but they can have unpredictable delays. In particular, a neighbour in G can be unpredictably long to appear in the contact list.

Solving the Leader Election Problem. We aim at a general distributed computability study of this model. We therefore introduce partial global knowledge in order to overcome the single sink condition of [CSS04] and we look for necessary and sufficient knowledge to solve a given problem. The leader election problem is a fundamental problem in distributed programming. It has also proved to be a good benchmark for distributed computability characterisation [YK89,YK96b,BCG$^+$96,BV99,GM02,CGM08].

Our Results. In this paper, we give a simple characterisation of the partial knowledge that enables to solve Election problem in networks with unknown participants. From our results, it appears that in the unknown participants model, the partial knowledge that a node can have initially about the structure of the underlying network has a dramatic impact.

As a consequence, we prove that knowing the size of the network enables to solve Election problem on every network whereas knowing only an upper bound on the size is not enough. We also prove that knowing the number of sink components in the initial graph is also a sufficient condition to solve it.

Related Work. The network with unknown participants model presented here is a slight variation of a model that has been formally introduced and studied in [CSS04]. In [CSS04], processes are endowed with a participant detector returning a set of initial out-neighbours. Initial values from such a participant detector determines the initial communication network G^0. Moreover, the communication networks end up being a complete graph by allowing direct communication as

soon as the network name of a node is learnt from a received message. In the model of this paper, the communication graph G becomes at most the undirected version of the initial graph G^0, but, computability is equivalent. Indeed, it is always possible to add a "routing layer" to build an end-to-end communication overlay in order to address a process which is not a neighbour. The meaning of "*knowledge*" is also different. In [CSS04], it is meant to correspond to the initial contact lists, that is G^0 with our notations. In this paper "knowledge" denotes the partial information that a node can have about the global structure.

In [CSS04], Cavin, Sasson and Schiper have investigated the Consensus Problem and showed a necessary and sufficient condition for computability of Consensus. They do not assume any partial knowledge about the initial graph and they show that the existence of at most one sink in the strongly connected components of the initial graph is both necessary and sufficient in order to solve Consensus. From a computability point of view, in this model, to the best of our knowledge, only the solvability of the Consensus Problem has been considered so far [CSS04]. Fault-tolerant versions of the Consensus problem were considered : in [GT07], the precise link in the unknown participant model between synchrony and fault-tolerance is given; in [ABFG08], byzantine faults are investigated; in [GSAS12], an eventually strong failure detector is presented.

So only [CSS04] seems to consider reliable networks with unknown participants. But going further in history, such studies for reliable communication networks, but with a partial knowledge, were actually introduced by Angluin [Ang80] in her seminal work for *anonymous networks*. The precise impact of some specific knowledge on distributed computability in anonymous networks has been thoroughly investigated by Yamashita and Kameda [YK96b,YK96a]. Boldi and Vigna have presented general computability results in [BV99].

We show here that, for Election problem, even without failure and with identities, there are still impossibility results. Our principal lemma used in proof for impossibility is a "Angluin's like" lemma called *Isolation Lemma*. It is extended to get a complete characterisation of which partial knowledge are sufficient and necessary to solve Election problem. It seems very likely that it is possible to leverage this lemma to get full computability results.

The problem of describing which arbitrary knowledge enables to solve the Election Problem was introduced in [GM02], where it is solved for a specific model. It has been solved for the standard message passing model in [CGM12]. These papers use quasi-simulation techniques introduced in [MMW97] but contrary to the model investigated in [YK96b,BV99,CGM12], it should be noted that unknown participants networks are communication networks where the difficulty arises from asynchrony and not from synchronous executions. It is therefore a qualitatively different model where computability mostly relates to termination detection and not to symmetry breaking.

The paper is organized as follows. First we present standard graph notation that we will use to describe formally the unknown participant model. We first present an algorithm that enables every node to compute the set of vertices one can reach from this node in the initial digraph. We then present the general

Isolation Lemma and derive our necessary condition on knowledge. Building on our first algorithm, we give an Election algorithm that proves that the necessary condition is actually sufficient. We conclude with some applications of our main theorem.

2 Graphs Properties and Reachable Vertices

Definitions are standard [RM00]. Let G be a directed graph (resp. undirected), where $V(G)$ is its set of vertices, and $E(G)$ is its set of arcs denoted (u, v) (resp. edges denoted $\{u, v\}$). A directed graph is called a digraph. We identify undirected graphs with symmetric digraphs and use graph and digraph interchangeably. Let *deg(v), the degree of the vertex v*. We denote by $pred(v) = \{u|(u, v) \in E(G)\}$ (resp. $next(v) = \{w|(v, w) \in E(G)\}$) the set of predecessors (resp. successors) of v. A directed *path* c (resp. an undirected path $\overline{c}$) linking u and v is a sequence of disjoint vertices $\{s_1, ...s_k\} \subseteq V(G)$ where for all $i < k$, $(s_i, s_{i+1}) \in E(G)$ (resp. $(s_i, s_{i+1}) \in E(G)$ or $(s_{i+1}, s_i) \in E(G)$), $s_1 = u$ and $s_k = v$. The *length of a path c*, denoted by $|c|$, is equal to the numbers of arcs composing it and the directed *distance d* (resp. undirected distance $\overline{d}$) between vertices is the length of the smallest directed (resp. undirected) path in G between u and v. A *strongly connected* (resp. connected) digraph is a digraph where the directed (resp. undirected) distance between any two vertices is always defined. We will consider only connected digraphs. Any digraph can be decomposed in strongly connected subgraphs (called components). A component that has no successor is called a *sink*. A vertex v is *reachable* from u in G if there is a directed path from u to v. We denote by $Reach_G(v)$, the *set of vertices reachable from v* in G.

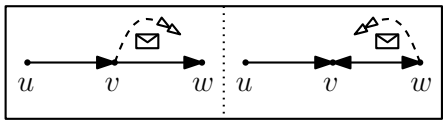
Remark 2.1. *The following propositions are equivalent:*

(i) G *is strongly connected.*
(ii) $\forall v, v' \in V, Reach_G(v) = Reach_G(v')$
(iii) $\forall v, v' \in V, v' \in Reach_G(v)$
(iv) $\forall v \in V, Reach_G(v) = V$

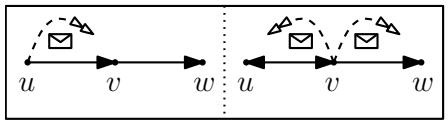
A graph H is a *subgraph* of a graph G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. A *labelled graph* $\mathbf{G} = (G, \lambda)$ is a graph G endowed with a labelling $\lambda : V \to \Lambda$ on its vertices or edges where Λ, is the set of labels. Note that if $\mathbf{H} = (H, \lambda_H)$ is a subgraph of $\mathbf{G} = (G, \lambda_G)$, then each node $v \in V(H)$ has the same label in $\mathbf{H}$ and in $\mathbf{G}$.

A *homomorphism* φ from H to G is a function $\varphi : V(H) \to V(G)$ such that for every $(u, v) \in E(H)$, there is $(\varphi(u), \varphi(v)) \in E(G)$. An *isomorphism* φ is a bijective homomorphism such that φ^{-1} is a homomorphism. A homomorphism (resp. isomorphism) φ from $\mathbf{G} = (G, \lambda_G)$ to $\mathbf{H} = (H, \lambda_H)$ is a homomorphism (resp. isomorphism) from G to H such that for each $v \in V(G)$, v and $\varphi(v)$ have the same label, i.e., $\lambda_G(v) = \lambda_H(\varphi(v))$.

Definition 2.2. *A subgraph* $\mathbf{H}$ *of* $\mathbf{G}$ *is a subgraph* closed by successors *of* $\mathbf{G}$*, denoted by* $\mathbf{H} \sqsubseteq\downarrow \mathbf{G}$*, if for every* $(u, v) \in E(G)$*, if* $u \in V(H)$ *then* $v \in V(H)$ *and* $(u, v) \in E(H)$*.*



(a) v sends a message. w receives it. After the reception, w can talk to v



(b) u sends a message. v receives it. After the reception, v can talk to both u and its initial neighbour w

Fig. 1: Different executions for the same initial graph.

This relation $\sqsubseteq\!\!\downarrow$ is extended for two arbitrary graphs $\mathbf{H}'$ and $\mathbf{G}$ where $\mathbf{H}'$ is isomorphic to a graph $\mathbf{H}$ and $\mathbf{H} \sqsubseteq\!\!\downarrow \mathbf{G}$. Note that if $\mathbf{H} \sqsubseteq\!\!\downarrow \mathbf{G}$ then for every $v \in V(H)$, $Reach_H(v) = Reach_G(v)$.

3 Model

The message passing model. A network is defined by a (possibly symmetric) digraph G, where $V(G)$ is the set of processes, and $E(G)$ is the set of communication channels. Processes communicate by sending and receiving messages via some ports. The communication channels linking ports between processes are asynchronous but reliable and $FIFO$.

Processes identities. Each process v is endowed with a unique label, $id_G(v)$, the identity of the process. We denote it by id_v if the context permits it. As there are several such labellings for a same graph (permutation, renaming), we consider all of them using an injective function $id_G : V \to \mathbb{N}$ giving a unique identity to every process. We denote by (G, id_G) such a graph and let $\mathcal{G}_{id}$ be the set of all connected graphs and their assignations of possible identities. For every family of graphs $\mathcal{F}$, we denote the family $\mathcal{F}_{id} = \{(G, id_G) | G \in \mathcal{F} \wedge (G, id_G) \in \mathcal{G}_{id}\}$.

Port labelling. Each process v can address its different neighbours using a bijective port numbering function $\delta_v : V \to \mathbb{N}$ giving a unique number to every port of v. When v receives a message from a neighbour w, it receives the message via the port $\delta_v(w)$. Since a process does not initially know all its neighbours, processes have access to a local variable denoted by CONTACTS containing the port numbers corresponding to their known neighbours. We explain its usage later. In order to ease notation, we consider a port labelling such that for all neighbours u, v in G, the port $\delta_u(v)$ corresponding to the channel linking u to v is denoted by $id_G(v)$, the identity of v. Therefore, CONTACTS contains the identities of known neighbours.

Graph labelling. The state of each process is represented by a label $\lambda(v)$ associated to the corresponding vertex $v \in V(G)$. Note that $\lambda(v)$ initially contains the identity of v and its known neighbours list CONTACTS_v cited above. Let $\mathbf{G} = (G, \lambda)$, such a labelled graph. For all arcs $(u, v) \in E(G)$, let $B(u, v)$ be the queue containing the messages in transit from u to v. Initially, $B(u, v)$ is empty for all arcs (u, v).

Distributed algorithm. We use the definition given by Tel in [Tel00] for distributed algorithms and executions. A distributed algorithm is a set of state transition rules. Such transition rules are function of the current local state of the system. In our setting, three kinds of transitions are possible for a process v: it can modify its state, it can receive a message from a neighbour or it can send a message to all its known neighbours. See Figure 1.

Let $(\lambda_v, in, m) \vdash (\lambda'_v, send, m')$, denote a recursive relation on state transition of a process v (the current state is function of the previous), with λ_v, the state of v before transition and λ'_v, its state after, m and m' are messages, in is the incoming port number (or $\perp$) and $send$ is either $\perp$ or all. When v modify its state, $in = send = \perp$ and $m = m' = \perp$; the state of v becomes λ'_v after this transition. When v receives a message m sent by u, $in = id_u$, $send = \perp$, $m \neq \perp$ and $m' = \perp$; the message m is in $B(u, v)$ and v receives m via the port id_u. If id_u is not known, v updates CONTACTS_v, the list of the neighbours it knows. The state of v becomes λ'_v after this transition. When v sends a message m', it sends the same message to all its known neighbours using a primitive SendAll. In this case, $in = \perp$, $send = all$, $m = \perp$ and $m' \neq \perp$; the message m' is added to all queues $B(v, w)$ such that $id_w \in \text{CONTACTS}_v$, and the state of v becomes λ'_v after this transition.

A distributed algorithm $\mathcal{A}$ in the message passing model is a set of algorithms $(\mathcal{A}_v)_{v \in V(G)}$ distributed over the nodes of the network. A transition of the algorithm is a transition of a process v according to its local algorithm $\mathcal{A}_v$.

Distributed algorithm execution representation. An execution ρ of a distributed algorithm $\mathcal{A}$ is a sequence of changes on vertices state. An execution is represented by a sequence of couples $[(\lambda^0, B^0), (\lambda^1, B^1), ..., (\lambda^n, B^n)]_\rho$ where, at step i, λ^i is the state of the system and B^i, the set of messages in transit. Let λ^i_v, the state of vertex v at step i. The initial state λ^0_v is the state of process v before the execution. A transition from step i to step $i+1$ is performed by one and only one process which executes a transition cited above in its local algorithm. This transition leads to the next state of v: λ^{i+1}_v and B^{i+1} and all other processes keep the same state as in λ^i. Note that any asynchronous execution (including the synchronous execution) can be represented this way.

Problem Specification. A specification of a problem has to describe the expected relations on the initial and final labelling of the graph where the problem has to be solved. As processes have to take a decision or compute some values in order to address a problem, we use a dedicated label. Let $out_G(v)$ be the label indicating the *decision computed* by a process v in G, denoted by $out(v)$ if the context permits. Note that a *final labelling* of a graph, denoted by $\mathbf{G}^{out} = (G, \lambda^{out})$ contains, for every process v, its final decision $out(v)$ ($\perp$ if the process has not decided any value). We define a *specification* as a *relabelling relation* $\mathcal{S}$ between initial labelled graphs and final labelled graphs.

Execution and Algorithm Properties. An execution *stabilises* if there is a step i_0 where no process can progress in its local algorithm and no message is in transit. In an execution, a process v *decides* if it eventually writes a value in *out* and if it does it only once during the execution. An execution terminates if

it stabilises and if every process decides. An algorithm terminates on (G, λ^{in}) with if every execution of the algorithm on (G, λ^{in}) terminates. In an execution ρ that terminates, each process v has an output value $out_v \neq \perp$; in this case, we say that out_v is the final label of v in ρ.

Let $\mathcal{S}$ be the specification of a problem. An execution ρ of $\mathcal{A}$ in a graph $(G, \lambda^{in}) \in \mathcal{G}^{in}$ satisfies the *Correction Property* if ρ terminates and (G, λ^{out}), the final labelling computed by ρ satisfies $(G, \lambda^{in})\mathcal{S}(G, \lambda^{out})$. An algorithm $\mathcal{A}$ is *valid for a specification* $\mathcal{S}$ in a graph $(G, \lambda^{in}) \in \mathcal{G}^{in}$ if every execution ρ satisfies the Correction Property. We will say that $\mathcal{A}$ solves $\mathcal{S}$ on (G, λ^{in}) in such a case.

Knowledge and Family. As we will see, some problems need additional global information or knowledge to be solved. This information about the underlying network (e.g. a bound on the size of the system) is inserted in the initial label. Consider a function κ that encodes an arbitrary knowledge. An algorithm $\mathcal{A}$ solves $\mathcal{S}$ with knowledge κ, if for all G, $\mathcal{A}$ solves $\mathcal{S}$ on $(G, \kappa(G))$. Equivalently we have that, for any $\alpha \in \kappa^{-1}(\mathcal{G}_{id})$, there exists an algorithm $\mathcal{A}_\alpha$ that solves $\mathcal{S}$ on the family $\mathcal{F} = \kappa^{-1}(\alpha)$.

Solving a problem with partial knowledge is simply, for any possible value α of knowledge, solving the problem within the family of networks whose knowledge value is α. Considering arbitrary families of labelled graphs enables to represent any initial knowledge: e.g. if the processes initially know the size n of the network, then in the corresponding family $\mathcal{F}^{(n)}$, for each $\mathbf{G} \in \mathcal{F}^{(n)}$ and each $v \in V(G)$, $n = |V(\mathbf{G})|$ is a component of the initial label of v.

Universal Algorithm. We say that an algorithm solving a specification on all graphs of $\mathcal{G}_{id}$ is a universal algorithm. An algorithm is $\mathcal{F}-universal$ if the algorithm solves $\mathcal{S}$ for all graphs of the family $\mathcal{F}$. Abusively, We will say that an algorithm $\mathcal{A}$ is $\mathcal{F} - universal$ if $\mathcal{A}$ is $\mathcal{F}_{id} - universal$.

An algorithm is not universal when it is not correct for all graphs but it can be $\mathcal{F}-universal$. So, for every problem without a *universal* algorithm, we look for the necessary and sufficient condition on the knowledge, *i.e.* on the family $\mathcal{F}$, such that the problem can be solved with a $\mathcal{F} - universal$ algorithm.

4 Cartography of Reachable Vertices

Reach or "Cartography of Reachable Vertices Problem" is the problem consisting, for every vertex v in the digraph G^0, to compute a graph isomorphic to subgraph induced by the network initially accessible from v. We denote by $\mathbf{G}_{|Reach(v)}$ such a subgraph of G^0. This problem is investigated because its solution will be used as the basis of our main Election Algorithm. Interestingly, it also admits a universal algorithm (Algorithm 1).

Description of the algorithm. We first introduce the variables used by the algorithm. Each process v initially knows id_v, its identity and $Succ_v = \{id_{v'} \mid v' \in next(v)\}$, the set of the ids of its neighbours in the network it initially knows. These variables are not modified during the execution of the algorithm. Our algorithm is a flooding algorithm where each node v eventually collects the value of $(id_u, Succ_u)$ for every process u.

Algorithm 1: REACH Algorithm.

Output: out, Graph induced by M_v

```
I:(Initial Procedure) begin
    Send <id, M> to all identities of vertices into CONTACTS;
R:(Receiving a message <id_u, M_u> from u:) begin
    if id_u ∉ CONTACTS or M_u \ M ≠ ∅ then
        M ← M ∪ M_u;
        CONTACTS ← CONTACTS ∪ {id_u} if id_u ∉ CONTACTS;
        Send <id, M> to all identities of vertices into CONTACTS;
    if View(M) = Covered(M) ∧ out = ⊥ then
        out ← C(M)_{|Reach(id_v)};
```

Each process v has also a variable CONTACTS_v containing the list of the ids of its neighbours it knows, either because it initially knows them, or because it receives a message from them. Initially $\text{CONTACTS}_v = Succ_v$ and CONTACTS_v is updated each time v receives a message from a neighbour u such that $id_u \notin \text{CONTACTS}_v$. Finally, each process v has a mailbox M_v containing pairs of the form $(id, Succ)$. Intuitively, the mailbox contains all the information v has about the network. Initially, $M_v = \{(id_v, Succ_v)\}$. When a vertex v sends a message to its neighbours, it always sends a message of the form $< id_v, M_v >$, i.e., it sends all the information it has on the network.

Our algorithm is a flooding algorithm described by two rules. Initially, each process applies the rule **I** to send its initial mailbox (containing only $(id_v, Succ_v)$) to all its neighbours. The rule **R** is executed whenever a process receives a message $< id_u, M_u >$ from a neighbour u. If the received mailbox provides new entries, then the process learns new information about the network and it updates its mailbox. Moreover, if id_u is not in CONTACTS, then id_u is added to CONTACTS. Then, if the process has learned new information (i.e., if its mailbox or CONTACTS has changed), it sends a copy of its new mailbox to all its neighbours.

Computing a map from a mailbox. In order to explain the rule allowing a process v to writes a value in out_v, we need to first explain how to use the content of a mailbox to construct a digraph similar to the network communication graph. To do so, we define three functions: $View$, $Covered$ and $\mathcal{C}$ as follows.

- $Covered(M) = \{id_v | (id_v, Succ_v) \in M\}$
- $View(M) = \{id_v | \exists (id_u, Succ_u) \in M \wedge id_v \in Succ_u\} \cup Covered(M)$
- $\mathcal{C}(M) = (V_\mathcal{C}, E_\mathcal{C})$ is a digraph such that $V_\mathcal{C} = View(M)$ and $E_\mathcal{C} = \{(id,\ id') \mid (id, Succ) \in M \text{ and } id' \in Succ\}$

With those functions, we can prove that if M_v contains the list of successors of every node in the network, the reconstructed graph is isomorphic to the initial communication graph. By construction, the following lemma is proved.

Lemma 4.1. $\mathcal{C}\left(\{(id_v, Succ_v) \mid v \in V(G)\}\right) \simeq \mathbf{G}$

During the execution of the algorithm, as long as $Covered(M_v) \neq View(M_v)$, v can detect that it has not yet received the initial information from all the processes. When $Covered(M_v) = View(M_v)$, v can reconstruct a graph from M_v and it is possible that $\mathcal{C}(M_v)$ is isomorphic to $\mathbf{G}$ but it is not necessary. However, we will show that in this case, $\mathbf{G}_{|Reach(v)} \sqsubseteq\downarrow \mathcal{C}(M_v)$, and consequently, v can compute $\mathbf{G}_{|Reach(v)}$ by performing a depth-first traversal of $\mathcal{C}(M_v)$ from id_v and v can decide this value. Note that a mailbox can satisfy the constraint $View(M) = Covered(M)$ several times; this is due to the asynchrony of communications. We will elaborate on this interesting property later.

Properties of the algorithm. In order to prove the termination property and the correction of the algorithm, we start by some lemmas on the properties about the content of the mailbox. First, since processes have unique identities, we get the following lemma.

Lemma 4.2 (bounded content)**.** *For every step i and process v, $\mathcal{C}(M_v^i)$ is a subgraph of $\mathcal{C}\left(\{(id_v, Succ_v) \mid v \in V(G)\}\right)$.*

The next lemma shows that there exists an increasing order on the mailbox content during an execution.

Lemma 4.3. *For every execution ρ of an algorithm, for every process v that executes a transition at step i, $M_v^i \subseteq M_v^{i+1}$, $\text{CONTACTS}_v^i \subseteq \text{CONTACTS}_v^{i+1}$ and v sends a message if and only if $M_v^i \subset M_v^{i+1}$ or $\text{CONTACTS}_v^i \subset \text{CONTACTS}_v^i$.*

Proof. Processes update their local state only when a message is received. Let $m =< id_u, M_u >$, the message received by v from u at step i. If $id_u \notin \text{CONTACTS}_v^i$, then id_u is added to CONTACTS_v^{i+1} (line 6). So, $\text{CONTACTS}_v^i \subset \text{CONTACTS}_v^{i+1}$ and v sends messages to its neighbours. If $M_u \setminus M_v^i \neq \emptyset$, then an update of M_v^i is operated by v, and $M_v^i \subset M_v^{i+1}$. After this update (procedure R at line 5), v will send messages to its neighbours. Otherwise, we get $M_u \subseteq M_v^i$ and $id_u \in \text{CONTACTS}_v^i$. Thus, v performs no action during this step. □

By Lemma 4.2, the mailbox's content can only take a finite number of values, and by Lemma 4.3, it is increasing during any execution. Since, by Lemma 4.3, messages are only sent when the content of a mailbox is modified, there is a step i where the algorithm stabilises. We show in the next lemma that eventually each process gathers all available information.

Lemma 4.4 (Reception)**.** *For all v, v', there is a step i where $(id_{v'}, Succ_{v'}) \in M_v^i$.*

Proof. We prove this lemma by an induction hypothesis on $\overline{d}(v, v')$. First, assume that $\overline{d}(v, v') = 1$. Consider a step $h \leq i$ such that $M_v^h = M_v^i$ and $M_v^{h-1} \neq M_v^h$. At step h, v sends its mailbox M_v^h to its neighbourhood. We distinguish two cases: either v knows v' at step h or not.

Case 1: $\mathbf{id_{v'} \in contacts^h(id_v)}$. Since $id_{v'} \in \text{CONTACTS}^h(id_v)$, v sends a message $< id_v, M_v^h >$ to v' at step h. Since the channel are reliables, there exists a step $j > h$ where v' receives $< id_v, M_v^h >$ and thus, $M_v^i = M_v^h \subseteq M_{v'}^j$.

Case 2: $\mathbf{id_{v'} \notin contacts^h(id_v)}$. Since $id_{v'} \notin \text{CONTACTS}^i(id_v)$ and $\overline{d}(v, v')$, $id_v \in \text{CONTACTS}^0(id_{v'})$. Since v' applies eventually the rule **I** of the algorithm, and

since the channels are reliable, there is a step $j > 0$ where v receives a message from v'. Since $id_{v'} \notin \text{CONTACTS}_v^h$, it implies that $j > h$. Thus, $M_v^i = M_v^h \subseteq M_v^j$. At step j, when v receives the message from v', the algorithm ensures that v' is added to CONTACTS_v and that a message $< id_v, M_v^j >$ is sent to all the known neighbours of v including v'. By the previous case, there exists a step j' such that $M_v^j \subseteq M_{v'}^{j'}$ and since $M_v^i \subseteq M_v^j$, we are done.

Suppose now that $\overline{d}(v, v') > 1$. Let w be a neighbour of v such that $\overline{d}(w, v') = \overline{d}(v, v') - 1$. From the case where $\overline{d}(v, v') = 1$, we know that there exists a step j such that $M_v^i \subseteq M_w^j$. By induction hypothesis, there exists a step j' such that $M_w^j \subseteq M_{v'}^{j'}$. So $M_v^i \subseteq M_{v'}^{j'}$. □

From Lemmas 4.3 and 4.4, there exists a step i such that for all v, v', $(id_{v'}, Succ_{v'}) \in M_v^i$. Thus, the condition on line 8 is eventually satisfied for every process, proving that *every execution of the algorithm terminates*. It remains to prove that when v decides a value out_v isomorphic to $\mathbf{G}_{|Reach(v)}$.

Lemma 4.5 (Correction)**.** *For every process v, if $View(M_v) = Covered(M_v)$ then for every $w \in Reach_G(v)$, $id_w \in Covered(M_v)$. Consequently, $\mathcal{C}(M_v) \sqsubseteq\downarrow \mathbf{G}$.*

Proof. By contradiction, let $w \in Reach_G(v)$ be the closest process to v in $Reach_G(v)$ such that $id_w \notin Covered(M_v)$. Let $w' \in Reach_G(v)$ be a predecessor of w belonging to a shortest path from v to w. By the choice of w, $id_{w'} \in Covered(M_v)$. Thus, $id_w \in Succ_{w'}$, we get $id_w \in Covered(M_v) = View(M_v)$, a contradiction. □

Now, from Lemmas 4.4 and 4.5 which prove that every process v decides $\mathbf{G}_{|Reach(v)}$, we can give a first theorem on the computability of the REACH Problem:

Theorem 1. *There is an universal algorithm for* REACH.

5 Isolation Lemma

In this section, we present an isolation lemma to prove impossibility results caused by isolated executions in a subset of the network. As it has proved for anonymous network to be the basis for all impossibility proofs [Ang80,Cha06], the isolation lemma is presented like a lifting lemma. Initially, a process v knows only its outgoing neighbourhood. Any other neighbour u of v cannot receive a message from v before v received a message from u. If $H \sqsubseteq\downarrow G$ and if all messages sent from processes in $V(G) \setminus V(H)$ to processes in $V(H)$ are arbitrary delayed (the communication is asynchronous), the processes of H are isolated from the rest of the network and execute the algorithm as if they were only in H, without discovering their neighbours outside H before deciding. If such an execution terminates, isolated processes decide a final value for an execution in H and not in G.

We introduce a new notation in order to represent an extended labelled graph with messages in transit. Let $\underline{\mathbf{H}} = (H,\ \lambda_H,\ B_H)$ where (H, λ_H) is a labelled

graph and B_H, its queues of messages. The relation $\sqsubseteq\!\!\downarrow$ is extended between such graphs as follow, $(H,\ \lambda_H,\ B_H) \sqsubseteq\!\!\downarrow (G,\ \lambda_G,\ B_G)$ if $(H, \lambda_H) \sqsubseteq\!\!\downarrow (G, \lambda_G)$ and for every $(u, v) \in E(H)$, $B_H(u, v) = B_G(u, v)$. First, we remark that by isomorphism, a labelling of a graph $\mathbf{G}$ induces a labelling for every subgraph $\mathbf{H}$ of $\mathbf{G}$. Such initial labellings are independent of the algorithm used and satisfies the relation $\sqsubseteq\!\!\downarrow$ between $\mathbf{H}$ and $\mathbf{G}$.

Remark 5.1 (Initialisation)**.** *For every digraphs G and H such that $H \sqsubseteq\!\!\downarrow G$, for every initial labelling λ_G^0 of G, there is a labelling λ_H^0 of H, such that $(H, \lambda_H^0, \emptyset) \sqsubseteq\!\!\downarrow (G, \lambda_G^0, \emptyset)$.*

Next, we prove that any step of an execution on a graph $\mathbf{H}$ can be executed on every graph $\mathbf{G}$ satisfying $\mathbf{H} \sqsubseteq\!\!\downarrow \mathbf{G}$.

Lemma 5.2 (One step of execution)**.** *For all $\underline{\mathbf{H}} = (H,\ \lambda_H,\ B_H)$ and $\underline{\mathbf{G}} = (G,\ \lambda_G,\ B_G)$, if $\underline{\mathbf{H}} \sqsubseteq\!\!\downarrow \underline{\mathbf{G}}$ then every transition $(\lambda_H, in, m) \vdash (\lambda'_H, send, m')$ executed on $\underline{\mathbf{H}}$ can be executed on $\underline{\mathbf{G}}$. The graphs $\underline{\mathbf{H}'} = (H', \lambda'_H, B'_H)$ and $\underline{\mathbf{G}'} = (G', \lambda'_G, B'_G)$ obtained after the transition satisfy $\underline{\mathbf{H}'} \sqsubseteq\!\!\downarrow \underline{\mathbf{G}'}$.*

Proof. We prove this lemma by constructing a similar execution in $\underline{\mathbf{H}}$ and $\underline{\mathbf{G}}$ which preserves the relation $\sqsubseteq\!\!\downarrow$. A step of the execution corresponds to a local transition of a process v. If a process v sends a message m' to its neighbours in H, since for all $v \in V(H)$, $\textsc{contacts}_H(v) = \textsc{contacts}_G(v)$, v can send the same message to the same nodes in G, and consequently, for all $(v, v') \in E(H)$, $B_G(v, v') = B_H(v, v')$. Since v ends up in the same state in H and in G, $\underline{\mathbf{H}'} \sqsubseteq\!\!\downarrow \underline{\mathbf{G}'}$. Suppose now that a message $m \in B_H(v', v)$ is received from a process $v' \in V(H)$ via a port in in H. Since $(v', v) \in E(H)$; $B_G(v', v) = B_H(v', v)$ and thus v can also receive the message m from v' in G. Since m is removed from both queues, $B_G(v', v) = B_H(v', v)$. Note that v ends up in the same state in H and in G. In particular, if v' was not known by v, then v can now communicate with v' in H and in G. Consequently, $(H', \lambda'_H, B'_H) \sqsubseteq\!\!\downarrow (G', \lambda'_G, B'_G)$. □

For any graphs $\mathbf{G}$ and $\mathbf{H}$ such that $\mathbf{H} \sqsubseteq\!\!\downarrow \mathbf{G}$, for any algorithm $\mathcal{A}$ and any execution ρ_H on $\mathbf{H}$, we can apply the previous lemma iteratively to construct an execution ρ_G on $\mathbf{G}$ such that only the vertices in $V(H)$ are active in ρ_G and they behave exactly like in ρ_H. In such a case, we say that ρ_H is $\sqsubseteq\!\!\downarrow$ −lifted on $\mathbf{G}$. When considering executions that terminate, we get the following lemma.

Lemma 5.3 (Isolation Lemma)**.** *For all $\mathbf{G}, \mathbf{H}$ such that $\mathbf{H} \sqsubseteq\!\!\downarrow \mathbf{G}$, for every execution ρ_H of any algorithm $\mathcal{A}$ on H which terminates, there is an execution ρ_G of $\mathcal{A}$ on G such that $\forall v \in V(H)$, $out_{\rho_H}(v) = out_{\rho_G}(v)$.*

6 Election Algorithm

In this section, we study the classical election problem, denoted by Elec: one and only one process has to decide leader and all others should decide follower.

We prove that contrary to the classical model, even if nodes have identities, it is not possible to solve Elec with a *universal algorithm*. Therefore, we give a necessary and sufficient condition that determines which additional knowledge

enables to solve ELEC. The impossibility proof uses a standard simulation technique, based on the Isolation Lemma. To show that it is a sufficient condition, we show how processes can avoid $\sqsubseteq\downarrow$-lifted executions by delaying their decision when they are provided some additional knowledge satisfying this condition.

A Necessary and Sufficient Condition. If we have isolated executions, we might get more than one processes elected. So the condition on knowledge below enables to somehow forbid disjoint isolated executions.

Definition 6.1 ($\mathbb{C}_{\text{ELEC}}$)**.** *A family of graphs $\mathcal{F}$ satisfies $\mathbb{C}_{\text{ELEC}}$ if for every graphs $\mathbf{G}, \mathbf{H_1}, \mathbf{H_2} \in \mathcal{F}$ such that $\mathbf{H_1}, \mathbf{H_2} \sqsubseteq\downarrow \mathbf{G}$, we have $V(H_1) \cap V(H_2) \neq \emptyset$.*

We first prove that $\mathbb{C}_{\text{ELEC}}$ is necessary,

Lemma 6.2. *If there is an $\mathcal{F}$-universal algorithm solving* ELEC *then $\mathcal{F}$ satisfies $\mathbb{C}_{\text{ELEC}}$.*

Proof. Let $\mathcal{F}$ a family of labelled graphs that does not satisfy $\mathbb{C}_{\text{ELEC}}$ and $\mathcal{A}$, a $\mathcal{F}$-universal algorithm solving ELEC. There are three graphs $\mathbf{H_1}$, $\mathbf{H_2}$ and $\mathbf{G}$ in $\mathcal{F}$ such that $\mathbf{H_1}, \mathbf{H_2} \sqsubseteq\downarrow \mathbf{G}$ and $V(H_1) \cap V(H_2) = \emptyset$. We build an execution ρ_G on G as follows: the Isolation Lemma can be applied for H_1 and G, thus ρ_G can begin by a $\sqsubseteq\downarrow$ −lifted execution on H_1. Since $\mathcal{A}$ is $\mathcal{F}$-universal and $H_1 \in \mathcal{F}$, there is one elected process v_1. As H_1 and H_2 are vertex-disjoint and Isolation Lemma can also be applied for H_2 and G, we can extend ρ_G by taking a second $\sqsubseteq\downarrow$ −lifted execution on H_2. Since $\mathcal{A}$ is $\mathcal{F}$-universal and $H_2 \in \mathcal{F}$, there is one elected process v_2 with $v_2 \neq v_1$ because $V(H_1) \cap V(H_2) = \emptyset$. At this step, the labelling of $\mathbf{G}$ is not valid for ELEC because there are two elected vertices and their decisions are final for the execution. □

To prove that $\mathbb{C}_{\text{ELEC}}$ is a sufficient condition, we propose a $\mathcal{F}$-universal algorithm for any family $\mathcal{F}$ satisfying $\mathbb{C}_{\text{ELEC}}$. The algorithm presented below is an extension of Algorithm 1.

Description of the algorithm. In the algorithm, a process v does not only broadcast $Succ_v$, but it also broadcasts the tuple $(id_v, M_v, status_v)$ where $status_v \in \{\text{LEADER}, \text{FOLLOWER}, \perp\}$ is the content of out_v. In such a way, a process u can detect whether v has been elected or not and it can learn what v knows about the other processes. To do so, each process v has now a "super" mailbox, denoted $\mathcal{V}_v$, containing tuples of the form $(id, M, status)$. When a process v sends a message to its neighbours, it always sends a message of the form $< id_v, \mathcal{V}_v >$ (i.e., it sends its super mailbox instead of its mailbox). This structure gives, to processes, a view of the local states of the known processes. In order to avoid some $\sqsubseteq\downarrow$ −lifted executions, we have to delay the decisions of the processes for a sufficiently long period using an additional knowledge. Before a process writes in *out*, it checks that the reconstructed graph is in $\mathcal{F}$; to do so, we assume that each process knows the *characteristic function* $\chi_{\mathcal{F}}$ of the family $\mathcal{F}$ (the additional knowledge). When called on $\mathcal{C}(M)$, this function returns true if $\mathcal{C}(M) \in \mathcal{F}$ and false otherwise. To prevent two processes to be elected at the same time, we add an additional condition related to the supermailbox of all known processes at line 9. This condition ensures that two processes that want to decide LEADER have to actually know the state of each other. Changes between the previous and

Algorithm 2: A $\mathcal{F}$-universal Election Algorithm

Input: $\chi_{\mathcal{F}}$, Characteristic function of $\mathcal{F}$

1 **I:**(Initial Procedure) **begin**
2 SendAll $<id, \mathcal{V}>$;
3 **R:**(Receiving a message $<id_u, \mathcal{V}_u>$ from u) **begin**
4 **if** $id_u \notin$ CONTACTS *or* $\mathcal{V}_u \setminus \mathcal{V} \neq \emptyset$ **then**
5 $M \leftarrow M \cup \bigcup_{\{M_w | \exists (id_w, M_w, status_w) \in \mathcal{V}_u\}} M_w$;
6 $\mathcal{V} \leftarrow \mathcal{V} \cup \mathcal{V}_u \cup \{(id, M, out)\}$;
7 CONTACTS $\leftarrow$ CONTACTS $\cup \{id_u\}$ if $id_u \notin$ CONTACTS ;
8 **if** $out = \perp \wedge \chi_{\mathcal{F}}(\mathcal{C}(M)) \wedge View(M) = Covered(M) \wedge \forall id_w \in View(M), \exists (id_w, M, status_w) \in \mathcal{V}$ **then**
9 **if** $id = \min\{id' \mid id' \in View(M)\} \wedge \nexists (id_w, M_w, \text{LEADER}) \in \mathcal{V}$ **then**
10 $out \leftarrow$ LEADER;
11 **else**
12 $out \leftarrow$ FOLLOWER;
13 $\mathcal{V} \leftarrow \mathcal{V} \cup \{(id, M, out)\}$;
14 **if** $\mathcal{V}$ *or* CONTACTS *have changed* **then**
15 SendAll $<id, \mathcal{V}>$;

the next algorithm are the content of the messages (line 2), the manipulation of the new structure (lines 5 and 6) and the condition of termination (lines 8 and 9) as seen above.

Properties of the algorithm. Consider an execution ρ_G of the algorithm on $\mathbf{G}$. Note that as in the previous algorithm M_v can only take a finite number of values and can only increase during the execution of the algorithm. Consequently, as before, there exists a step i where all processes have the same mailbox $M = \{(id_v, Succ_v) \mid v \in V(G)\}$.

The following lemma is immediate and ensures that for each process v, $\mathcal{V}_v$ can only take a finite number of values.

Lemma 6.3. *For every process v, for every step i, for every $(id_w, M, status) \in \mathcal{V}_v^i$, there is a step $i' \leq i$ such that $M_w^{i'} = M$ and $out_w^{i'} = status$.*

Similarly as for mailboxes in the previous algorithm, we can show that for each step i, $\mathcal{V}_v^i \subseteq \mathcal{V}_v^{i+1}$ and v sends some messages at step i if and only if $\mathcal{V}_v^i \subsetneq \mathcal{V}_v^{i+1}$. Since for every v, the content of $\mathcal{V}_v$ is increasing and can only take a finite number of values, the execution stabilises. An eventually, each process knows the state of all other processes. The proof is similar to the proof of Lemma 4.4.

Lemma 6.4. *For every processes v, v', for every step i, there is a step $i' > i$ such that $(id_v, M_v^i, out_v^i) \in \mathcal{V}_{v'}^{i'}$.*

Consequently, there exists a step where all processes have the same super-mailbox. Note that in such a step, all processes have the same mailbox $M = \{(id_v, Succ_v) \mid v \in V(G)\}$, and for all $v, w \in V(G)$, there exists $(id_v, M, status) \in$

$\mathcal{V}_w$. Note that in this case, the condition on line 8 is satisfied and thus, eventually, each process decides a value. Therefore, the execution terminates. It remains to show that in each execution, there is always one and only one elected process.

Lemma 6.5. *For every execution, there is at least one elected process.*

Proof. Let id_{min} be the minimum identity present in the network and let v be the unique process such that $id_v = id_{min}$. Since every process eventually decides, there is a step i where the state of v satisfies the condition on line 8. We consider two cases. Either there exists some $(id_w, M_v, \text{LEADER}) \in \mathcal{V}_v^i$ and by Lemma 6.3, w has been elected and the lemma is proved. Or, there is no $(id_w, M_v, \text{LEADER}) \in \mathcal{V}_v^i$ and then v writes LEADER in $out(v)$. In both cases, there is always at least one elected process. □

Lemma 6.6. *For every execution ρ, there is at most one elected process.*

Proof. Suppose that there are two processes u and v elected at step i and j in an execution ρ. Wlog, we assume that $i \leq j$. Let $H_u = \mathcal{C}(M_u^i)$ and $H_v = \mathcal{C}(M_v^j)$. Since u and v are elected respectively at step i and j, $View(M_u^i) = Covered(M_u^i)$, $H_u \in \mathcal{F}$, $View(M_v^j) = Covered(M_v^j)$ and $H_v \in \mathcal{F}$. Consequently, $H_u \sqsubseteq\!\downarrow G$ and $H_v \sqsubseteq\!\downarrow G$. Two cases are possible when u is elected at step i: either $id_v \in View(M_u^i)$, or not.

Case 1: $\mathbf{id_v} \in \mathbf{View(M_u^i)}$. In this case, line 9 ensures that $id_u < id_v$. Moreover, lines 8 and 9 and Lemma 6.3 ensure that there is a step $h < i$ such that $M_v^h = M_u^i$ and $out_v^h = \perp$. Since $id_u \in View(M_u^i) = View(M_v^h)$, and since M_v can only increase during the execution (recall that $h < i < j$), $id_u \in View(M_v^j)$. Consequently, the condition on line 9 is not satisfied by the state of v at step j, and v is not elected in this case.

Case 2: $\mathbf{id_v} \notin \mathbf{View(M_u^i)}$. Since $H_u \sqsubseteq\!\downarrow G$ and $H_v \sqsubseteq\!\downarrow G$ and since $\mathcal{F}$ satisfies $\mathbb{C}_{\text{ELEC}}$, there exists w such that $id_w \in V(H_u) \cap V(H_v)$. By Lemma 6.3 and since $id_w \in V(H_u)$, we know that there is a step $i' \leq i$ where $M_w^{i'} = M_u^i$. Similarly, we know there exists a step $j' \leq j$ where $M_w^{j'} = M_v^j$. Note that $id_v \notin View(M_w^{i'})$ and that $id_v \in View(M_w^{j'})$. By Lemma 4.3, it implies that $i' < j'$ and that $M_u^i = M_w^{i'} \subsetneq M_w^{j'} = M_v^j$. Therefore $id_u \in View(M_v^j)$ and since v is elected at step j, there is some $(id_u, M_v^j, status) \in \mathcal{V}_v^j$. By Lemma 6.3, it implies that there exists a step $i'' < j$ such that $M_u^{i''} = M_v^j$ and $out_u^{i''} = status$. Since $id_v \in View(M_u^{i''})$ and $id_v \notin View(M_u^i)$, Lemma 4.3 implies that $i < i''$. Since $out_u^{i''} = out_u^{i+1} = \text{LEADER}$, $(id_u, M_v^j, \text{LEADER}) \in \mathcal{V}_v^j$. Thus, the condition on line 9 is not satisfied by the state of v at step j, and v is not elected in this case. □

Consequently, any execution of Algorithm 2 terminates and leads to one elected process if $\mathcal{F}$ satisfies $\mathbb{C}_{\text{ELEC}}$. Together with Lemma 6.2, we get

Theorem 2. *There is an $\mathcal{F}$-universal algorithm for* ELEC *if and only if $\mathcal{F}$ satisfies $\mathbb{C}_{\text{ELEC}}$.*

Applications. As first example, given $n \in \mathbb{N}$, consider the family of graphs with n vertices, denoted by $\mathcal{G}^{(n)}$. As every strict subgraph H of a graph $G \in \mathcal{G}^{(n)}$ has strictly less than n vertices, $H \notin \mathcal{G}^{(n)}$ and $\mathcal{G}^{(n)}$ trivially satisfies $\mathbb{C}_{\text{ELEC}}$. It is also possible to directly design an Election algorithm by simply waiting until *Covered* is of size n. Another, maybe less obvious, example where ELEC is possible is the family of graphs with n sink components, denoted by $\mathcal{P}^{(n)}$. This family satisfies $\mathbb{C}_{\text{ELEC}}$ because every two subgraphs H, H' of a graph $G \in \mathcal{P}^{(n)}$ have to share the n sink components if they also belong to $\mathcal{P}^{(n)}$. Thus, H and H' can not be disjoint.

Note that in [CSS04], the authors consider only families that are closed by $\sqsubseteq\!\downarrow$. Such families satisfy $\mathbb{C}_{\text{ELEC}}$ if and only if their graphs have only one sink. In this case, we obtain for ELEC a "one sink" condition similar to the one given in [CSS04].

Families defined by having a bound on the size are also closed by $\sqsubseteq\!\downarrow$. The corresponding families contains graphs with two sinks and it is therefore impossible to elect knowing a bound. An integer k is a *tight bound* for the size of G if $|V(G)| \leq k < 2|V(G)|$. Given $k \in \mathbb{N}$, the family $\mathcal{B}^{(k)}$ of graphs with a tight bound k admits an Election algorithm because $\mathcal{B}^{(k)}$ satisfies $\mathbb{C}_{\text{ELEC}}$. Indeed, consider $\mathbf{G}, \mathbf{H}, \mathbf{H'} \in \mathcal{B}^{(k)}$, k being a tight bound for $\mathbf{H}$, $\mathbf{H'}$, and $\mathbf{G}$ implies that $|V(H)| + |V(H')| > |V(G)|$. So when $\mathbf{H} \sqsubseteq\!\downarrow \mathbf{G}, \mathbf{H'} \sqsubseteq\!\downarrow \mathbf{G}$, we get $\mathbf{H} \cap \mathbf{H'} \neq \emptyset$. This majority argument also applies to the family of graphs where a tight bound is known for the number of sinks, so ELEC is also solvable in this case.

One major consequence of Theorem 2 is that ELEC cannot be solved on $\mathcal{G}_{id}$, and, furthermore, there is no maximum family, *i.e.* maximum knowledge, for which ELEC is solvable. Given any graph $\mathbf{G}$ with subgraphs $\mathbf{H_1}, \mathbf{H_2}$ such that $\mathbf{H_1}, \mathbf{H_2} \sqsubseteq\!\downarrow \mathbf{G}$, and $V(H_1) \cap V(H_2) = \emptyset$, the families $\{\mathbf{G}, \mathbf{H_1}\}$ and $\{\mathbf{G}, \mathbf{H_2}\}$ are incomparable and ELEC is solvable on both, whereas this problem has no solution on their union $\{\mathbf{G}, \mathbf{H_1}, \mathbf{H_2}\}$.

7 Conclusion

We investigated the computability of Election problem in the unknown participants model introduced in [CSS04]. Our result gives a simple condition on the partial knowledge that has to be provided to processes in order to solve this problem. This condition extends and improves the previous results known for the model of reliable unknown participants.

Before obtaining a general computability result, it is already possible to see that some other problems can be investigated with the same tools. For example, the $k-$Consensus can be solved with a similar algorithm. We do not give a proof but the condition on knowledge would be to forbid more than k disjoint $\sqsubseteq\!\downarrow$ $-$subgraphs having the same knowledge value as the whole graph.

An interesting open problem is to consider unknown participants in anonymous networks. The conditions given in this paper would remain true. But, from [BV01], it is expected that additional conditions will be necessary to overcome specific impossibilities related to anonymous networks.

References

[ABFG08] E. A. P. Alchieri, A. N. Bessani, J. Fraga, and F. Greve. Byzantine consensus with unknown participants. In *OPODIS*, page 22–40, 2008.

[Ang80] D. Angluin. Local and global properties in networks of processors. *STOC*, 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. 4th Israeli Symp. on Theory of Computing and Systems*, pages 16–26, 1996.

[BV99] P. Boldi and S. Vigna. Computing anonymously with arbitrary knowledge. In *PODC*, pages 181–188, 1999.

[BV01] P. Boldi and S. Vigna. An effective characterization of computability in anonymous networks. In *DISC*, 2001.

[CGM08] J. Chalopin, E. Godard, and Y. Métivier. Local terminations and distributed computability in anonymous networks. In *DISC*, 2008.

[CGM12] J. Chalopin, E. Godard, and Y. Métivier. Election in partially anonymous networks with arbitrary knowledge in message passing systems. *Distrib. Comput.*, 2012.

[Cha06] J. Chalopin. *Algorithmique distribuée, calculs locaux et homorphismes de graphes*. PhD thesis, Université Bordeaux 1, 2006.

[CSS04] D. Cavin, Y. Sasson, and A. Schiper. Consensus with unknown participants or fundamental self-organization. *Proc. of 3rd ADHOC-NOW*, July 2004.

[GM02] E. Godard and Y. Métivier. A characterization of families of graphs in which election is possible (*ext. abstract*). In *FOSSACS*, 2002.

[GSAS12] F. Greve, P. Sens, L. Arantes, and V. Simon. Eventually strong failure detector with unknown membership. *The Computer Journal*, 55(12):1507–1524, 2012.

[GT07] F. Greve and S. Tixeuil. Knowledge connectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks. In *DSN 2007*, pages 82–91, 2007.

[MMW97] Y. Métivier, A. Muscholl, and P. A. Wacrenier. About the local detection of termination of local computations in graphs. In *SIROCCO*, 1997.

[RM00] K.H. Rosen and J.G. Michaels. *Handbook of Discrete and Combinatorial Mathematics*. 2000.

[Tel00] G. Tel. *Introduction to Distributed Algorithms*. Cambridge U.P., 2000.

[YK89] M. Yamashita and T. Kameda. Electing a leader when processor identity numbers are not distinct. In *WDAG*, volume 392, pages 303–314, 1989.

[YK96a] M. Yamashita and T. Kameda. Computing on anonymous networks. I. characterizing the solvable cases. *IEEE TPDS*, 7:69–89, Jan 1996.

[YK96b] M. Yamashita and T. Kameda. Computing on anonymous networks. II. decision and membership problems. *IEEE TPDS*, 7:90–96, Jan 1996.