Distrib. Comput. (2010) 23:177–196
DOI 10.1007/s00446-010-0115-3

On the power of synchronization between two adjacent processes

Jérémie Chalopin · Yves Métivier

Received: 23 March 2009 / Accepted: 1 July 2010 / Published online: 24 July 2010

Abstract We study the power of local computations on labelled edges (which allow two adjacent vertices to synchronize and to modify their states simultaneaously in function of their previous states) through the classical election problem. We characterize the graphs for which this problem has a solution. As corollaries we characterize graphs which admit an election algorithm for two seminal models: Angluin's model and asynchronous systems where processes communicate with synchronous message passing (i.e., there is a synchronization between the process sending the message and the one receiving it).

1 Introduction

Various models of local computations on labelled graphs provide an elementary and convenient framework to define, to study and to teach basic algorithmic problems of distributed computing. In these models we have at our disposal only bare synchronization primitives and the corresponding local computations steps. It turns out that for many basic algorithmic problems arising in distributed computing such simple models are sufficient, and they allow either to formulate an algorithm solving the problem or to show formally that the problem is not solvable in the given context.

The relative simplicity of the local graph computation models facilitates the conception of algorithms and suitable combinatorial structures which subsequently can be translated and applied in a more realistic but also more involved setting. Such models give us an insight which is difficult to obtain in more elaborate models. Local graph computations often allow to delimit precisely the borderline between positive and negative results in distributed computing. It is clear that the possibility of solving a particular problem for a given class of networks depends on the power of the synchronization primitives and on the initial knowledge available to the computational agents. Better comprehension of all these factors enhances our understanding of basic distributed algorithmic problems.

As it is well established in the domain of distributed computing, some algorithmic problems like election, naming, termination detection, network topology recognition constitute basic building blocks for many other algorithms. In this work, characterizations of graphs in which election or naming are possible are obtained under two different models: local computations on (open) labelled edges.

As corollaries of our results, we obtain (for the first time to our knowledge) the characterization of graphs in which election is possible under two well-known models that are more realistic. More specifically, we show that the well known Angluin's model[1] of distributed computations of processors communicating via named channels can be reduced to our model of local computations on labelled edges. From this, we obtain a characterization of graphs where leader election can be solved in Angluin's model.

This work was supported by grant No ANR-06-SETI-015-03 awarded by Agence Nationale de la Recherche.

J. Chalopin
LIF, CNRS & Aix Marseille Université, 39 rue Joliot-Curie,
13453 Marseille, France
e-mail: jeremie.chalopin@lif.univ-mrs.fr

Y. Métivier (✉)
Université de Bordeaux, LaBRI UMR CNRS 5800,
351 cours de la Libération, 33405 Talence, France
e-mail: metivier@labri.fr

[1] Inspired by Milne and Milner's Model for distributed systems [22].

In an asynchronous system where processes communicate with synchronous message passing, each send event and the corresponding receive event happen simultaneaously (see [27, p. 47]). In other words, the sender and the receiver have to synchronize in order to be able to communicate. It has been defined by Hoare in [18]. As it is explained in [14, p. 40], examples of synchronous message passing are also given by Ada Rendezvous, Remote Procedure Calls and Remote Method Invocation. We show that this model can be reduced to the model of local computations on open labelled edges we study in this paper. From this, we also obtain a characterization of graphs where leader election can be solved in this model.

We also believe that local computations on edges enable a unified presentation of models like those presented above and recent models like population protocols [4] we discuss in Sect. 10. Furthermore, it seems to be much simpler and easier for manipulating and analyzing, and it enables comparisons of their power.

1.1 Our models

We consider networks of processors with arbitrary topology. A network is represented as a simple connected, undirected graph $G = (V(G), E(G))$. As usual the vertices represent processors and edges direct communication links. The state of each processor (resp. each link) is represented by the label $\lambda(v)$ of the corresponding vertex v (resp. by the label $\lambda(e)$ of the corresponding edge e). An elementary computation step will be represented by relabelling rules of the form given schematically in Fig. 1. If in a graph G there is a vertex labelled X linked to a neighbour labelled Z by an edge labelled Y then applying the rule R of Fig. 1, we replace X by a new label X', Y by a new label Y' and Z by a new label Z'; sometimes R will be denoted by: $R = ((X, Y, Z), (X', Y', Z'))$. The labels of all the other vertices are irrelevant for such a computation step and remain unchanged. The vertices of G changing the labels will be called *active* (and filled with black in figures). The computations using uniquely this type of relabelling rules are called in our paper *local computations on labelled edges*.

We consider also a variant of this model presented in Fig. 2. In this case the label of a vertex which appears in the rule does not change. The neighbour vertex used to match the rule is called *passive* (and marked as unfilled in figures).

Fig. 1 Graphical form of a relabelling rule R on labelled edges where $X' = f_1(X, Y, Z)$, $Y' = f_2(X, Y, Z) = f_2(Z, Y, X)$, $Z' = f_3(Z, Y, X)$, f_1, f_2 and f_3 are transition functions on triple of states; sometimes R is also denoted by $R = ((X, Y, Z), (X', Y', Z'))$

Fig. 2 Graphical form of a relabelling rule R on open labelled edges where $X' = f_1(X, Y, Z)$, $Y' = f_2(X, Y, Z) = f_2(Z, Y, X)$, and f_1, and f_2 are transition functions on triple of states; sometimes R is also denoted by $R = ((X, Y, Z), (X', Y', Z))$

Computations using this type of relabelling rules are called *local computations on open labelled edges*.

In both cases, all the other vertices of G not participating in such elementary relabelling step are called *idle*.

Thus a distributed algorithm in our model is simply given by some (possibly infinite but always recursive) set of rules of the type presented in Fig. 1 (resp. Fig. 2). A run of the algorithm consists in applying the relabelling rules specified by the algorithm until no rule is applicable, which terminates the execution. The relabelling rules are applied asynchronously and in any order, which means that given the initial labelling usually many different runs are possible. The distributed aspect comes from the fact that two consecutive non-overlapping steps may be applied in any order and in particular in parallel. The formal definitions of the model follow in Sect. 3.

Remark 1 Labels (states) are attached to vertices and edges. They make it possible to encode many different situations. If the network is anonymous then all vertices have the same label; vertices having unique identities, a distinguished vertex or any intermediate situation (partially anonymous) are other examples of labels attached to vertices; marks that encode a spanning tree is an example of labels attached to edges.

Remark 2 A rule of the type described in Fig. 1 enables to break the symmetry between two adjacent vertices.

1.2 Election and naming

In this paper, we focus on two classical problems of distributed computing that are election and naming. The election problem is one of the paradigms of the theory of distributed computing. A distributed algorithm solves the election problem if it always terminates and in the final configuration exactly one process is marked as *elected* and all the other processes are labelled *non-elected*. Moreover, it is supposed that once a process becomes *elected* or *non-elected* then it remains in such a state until the end of the algorithm. Election algorithms constitute a building block of many other distributed algorithms. The elected vertex acts as coordinator, initiator, and more generally performs some special role (cf. [26, p. 262]). If processes have initially unique identifiers, it is always possible to solve this problem by electing the process with the smallest identifier. Nevertheless, if we consider *anonymous* networks where processes do not have identifiers and execute the same algorithm, it is not always possible to solve the election problem. One aim of this paper

is to present a characterization of networks where this problem can be solved in our models.

The aim of a naming algorithm is to arrive at a final configuration where all processors have unique identities. Again this is an essential prerequisite to many other distributed algorithms which work correctly only under the assumption that all processors can be unambiguously identified. The enumeration problem is a variant of the naming problem. The aim of a distributed enumeration algorithm is to attribute to each network vertex a unique integer in such a way that this yields a bijection between the set $V(G)$ of vertices and $\{1, 2, \ldots, |V(G)|\}$.

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

1.3 Overview of our results

We recall that at each step of computation, labels are modified on exactly two endvertices linked by an edge and on this edge of the given graph, according to certain rules depending only on the label of this edge and on the labels of the two endvertices. Under this hypothesis, we give a complete characterization of labelled graphs for which there exists an election algorithm. More precisely, we prove that, given a simple labelled graph **G** (without self-loop or multiple edges) there exists an election algorithm for **G** if and only if **G** is minimal for the covering relation. First we consider multigraphs: graphs having possibly multiple edges without self-loop. For this class of graphs, a labelled graph **H** is a covering of a labelled graph **K** if there exists a label-preserving surjective homomorphism φ from H onto K such that, for every vertex v, the restriction of φ to the set of edges incident to v is a bijection between this set of edges and the set of edges incident to $\varphi(v)$. We then apply our result to characterize graphs which admit an election algorithm in Angluin's model.

Examples of minimal graphs includes trees, rings with a prime number of vertices or complete graphs. It shows that there exists many graphs where election can be solved in the models considered in this paper, while it is not possible in the model where processes communicate by asynchronous message passing. Indeed, thanks to the characterization given by Yamashita and Kameda [28,29] for asynchronous message passing systems where election can be solved, we know that in their model, election cannot be solved in rings or complete graphs and that there are some trees that does not admit an election algorithm. Moreover, the view-based techniques of Yamashita and Kameda cannot be adapted easily to obtain a characterization for local computations on labelled edges, or for Angluin's model. In fact, the views do not allow to capture easily the non-determinism of the execution that necessarily exists in our model, or Angluin's model.

The necessary condition we use is a slight generalization from Angluin's impossibility result [3] based on coverings. We also establish constructively the sufficiency of this condition. We use techniques inspired by the work of Mazurkiewicz [20] in a model described below (Sect. 1.4) these techniques have been already used to improve some results in an asynchronous message passing system [8] and seem complementary to techniques based on views.

We also consider the complexity of our algorithm: we show that both the number of steps of any execution of the algorithm and the size of the memory used by each process are polynomial (in the size of the graph), while Yamashita and Kameda's algorithm needs each process to have a memory of exponential size and to exchange messages of exponential size.

In the second part of this paper, we consider local computations on open labelled edges: at each step of computation labels are modified on exactly one edge and one endvertex of this edge of the given graph, according to certain rules depending on the label of this edge and the labels of its endvertices only.

We prove that this model is equivalent to the model studied in the first part by using a simulation algorithm. This result is not obvious: for example, using the first model, it is easy to give a name to each edge of a given graph such that for a given vertex v, all the edges incident to v have a different name; if we do not use the simulation algorithm this result is not trivial in the context of the second model.

As a consequence of this equivalence, we obtain a characterization of labelled graphs which admit an election algorithm in an asynchronous network with synchronous message passing (atomic receive/send).

In the last section we discuss the importance of the edge labels.

1.4 Related works: comparison and comments

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

Various local computation models studied in the literature are characterized by the relabelling rules that they use. Figure 4 presents schematically such rules and their hierarchy in terms of the computational power. Characterizations of graphs where naming and election can be solved exist for

each of these models, except for the models (3) and (4) that are studied in this paper.

Yamashita and Kameda [28] consider the model where, in each step, one of the vertices, depending on its current label, either changes its state, or sends/receives a message via one of its ports. Given a graph G they ask that there exists an election algorithm for every port numbering δ of G (a port numbering gives local names to incident edges). They proved that there exists an election algorithm for G if and only if the symmetricity of G is equal to 1, where the symmetricity depends on the number of vertices having the same view. The view from a vertex v of a graph G with a port numbering δ is an infinite labelled tree rooted in v obtained by considering all labelled walks in (G, δ) starting from v. This asynchronous message passing model is strictly less powerful than the model (6) in Fig. 4 but its computational power is not comparable with the computational power of the models (1), (2), (3), (4), (5) in Fig. 4. In [29], Yamashita and Kameda study the importance of the port labelling for the election problem in this message passing model. From the results of Boldi et al. [5], one can obtain different characterizations for the different models considered in [29], based on fibrations and coverings of directed graphs. In [8], Chalopin and Métivier present a new algorithm in this model. This algorithm is totally asynchronous and it needs a polynomial number of messages of polynomial size whereas previous election algorithms in the asynchronous message passing model (like [29]) are pseudo-synchronous and use messages of exponential size.

Mazurkiewicz [20] considers the asynchronous computation model where in one computation step labels are modified on a subgraph consisting of a node and its neighbours, according to rules depending on this subgraph only. This is the model (7) of Fig. 4. Mazurkiewicz's characterization of the graphs where enumeration/election are possible is based on the notion of unambiguous graphs and may be formulated equivalently using coverings of simple graphs [16, p. 256]. He gives a nice and simple enumeration algorithm for the graphs that are minimal for the covering relation, i.e., which can cover only themselves.

The characterization presented in this paper for local computations on labelled edges is not equivalent to the characterization of Mazurkiewicz. If we consider the ring with 4 vertices, denoted R_4, then it is minimal for the covering relation as defined in [16] (based on neighbours) but it is not minimal for the generalization given in this paper (based on incident edges). Indeed, for the generalization, R_4 covers the graph H defined by 2 vertices having a double edge (see Fig. 3).

Thus there exists an election algorithm for R_4 in the model of Mazurkiewicz and there does not exist an election algorithm for R_4 in the model studied in this paper.

Boldi et al. [5] consider a model where the network is a directed multigraph G. They consider models where the arcs can be labelled or not, and synchronous or interleaved activation models (in this paper we are interested in the interleaved model). When a processor is activated, it changes its state depending on its previous state and on the states of its ingoing neighbours; the outgoing neighbours do not participate in such an elementary computation step. From this work, we can easily deduce characterizations of the graphs where election is possible for the models (5) and (6) of Fig. 4.

From results obtained in this paper and [5] we deduce that the model (6) has a strictly greater power than the models (3) and (4). It turns out that for all these three models, election and naming over a given graph G are equivalent. We should note that the model (4) was also examined by Mazurkiewicz [21] who gives a characterization based on equivalence relations over graph vertices and edges. Let us note by the way that although the model (1) and the model (3) seem to be very close, the graphs for which the naming problem and the election problem can be solved are very different for both models. The intuitive reason is that if we allow to label the edges as in (3) then each processor can subsequently consistently identify the neighbours. On the other hand, in the model (1) where edges are not labelled, a vertex can never know if it synchronizes with the same neighbour as previously or with another one.

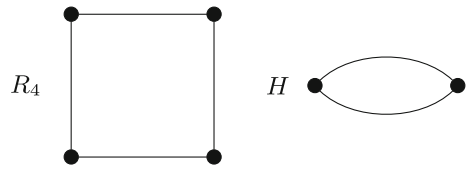


Fig. 3 The graph R_4 covers the graph H

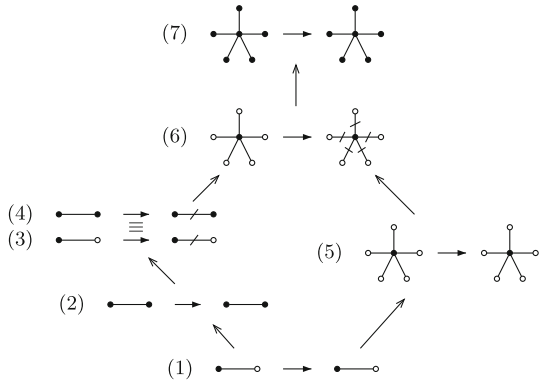


Fig. 4 A hierarchy of local computations models described by the different kinds of relabeling rules they use. The model hierarchy is displayed as follows. We write $(i) \rightarrow (j)$ for two models (i) and (j) if (j) can simulate (i) but not vice versa. This means that (j) has a greater computational power than (i); this relation is transitive. We write $(i) \equiv (j)$ if (i) and (j) have the same computational power. The computational power of model (5) is incomparable with the power of models (2), (3) and (4). A mark on an edge means that edges are labelled (and labels can be changed). We recall the presentation convention: a vertex filled with black can change its state while an unfilled vertex cannot change its state

In [9], the model (1) of Fig. 4 is studied: in one computation step a vertex modifies its state accoding to its state and the state of one of its neighbours. In this model, the graphs admitting a naming algorithm are submersion-minimal graphs. (A graph G is a submersion of a graph H via a homomorphism γ if for each $v \in V(G)$, γ is surjective on the neighbourhood $N_G(v)$, that is $\gamma(N_G(v)) = N_H(\gamma(v))$). In this model, the election and the enumeration problem are not equivalent. The characterization of graphs admitting an election algorithm is also given but it is more involved.

In [6], the model (2) of Fig. 4 is studied: in one computation step, two neighbours modify simultaneously their labels according only to their states (the edges are not labelled). In this model, the graphs admitting both naming and election algorithms are pseudo-covering-minimal graphs. (A graph G is a pseudo-covering of H if there exists a homomorphism φ from G onto H and a partial graph G' of G such that G' is a covering of H via the restriction of φ to G'.)

The study of local computations uses various locally constrained graph homomorphisms. Some properties and a complexity classification may be found in [10,12,13].

1.5 Summary

Section 2 reviews basic notions of graphs and labelled graphs. Section 3 presents local computations on labelled edges. Section 4 is devoted to the problems of election and enumeration. Coverings are presented and impossibility results are given. In Sect. 5, an enumeration algorithm using local computation on labelled edges is described: it enables to characterize the graphs for which there exists an election algorithm; we prove it and we give an analysis. Section 6 presents and studies Angluin's model. Section 7 proves the equivalence between local computations on labelled edges and local computations on open labelled edges. Section 8 characterizes graphs which admit an election algorithm in a synchronous message passing system. Sections 9 and 10 compare the power of the different models presented in this paper. Section 11 is the conclusion.

This paper is an improved version of the extended abstract [7].

2 Basic notions and notation

2.1 Graphs

The notation used here is essentially standard [25]. We consider finite, undirected, connected graphs without self-loop having possibly multiple edges. If $G = (V(G), E(G), \text{Ends})$ is a graph, then $V(G)$ denotes the set of vertices, $E(G)$ denotes the set of edges and Ends denotes a map assigning to every edge a set of two vertices: its ends. Two vertices u and v are said to be adjacent or neighbours if there exists an edge e such that $\text{Ends}(e) = \{u, v\}$. In this paper, graphs may have several edges between the same two vertices; such edges are called multiple edges. A simple graph $G = (V(G), E(G))$ is a graph with no self-loop nor multiple edges: $E(G)$ can be seen as a set of pairs of different vertices of G.

Let e be an edge, if the vertex v belongs to $\text{Ends}(e)$ then we say that e is incident to v and v is an endvertex of e. The set of all the edges of G incident with v is denoted by $I_G(v)$. The set of neighbours of v in G, denoted by $N_G(v)$, is the set of all vertices of G adjacent to v. For a vertex v, we denote by $B_G(v)$ the ball of radius 1 with center v, that is the graph with vertices $N_G(v) \cup \{v\}$ and edges $I_G(v)$.

The degree of a vertex v in a graph G, denoted by $deg_G(v)$, is the number of edges incident with v, it is equal to the cardinality of $I_G(v)$.

A homomorphism between G and H is a mapping $\gamma\colon V(G) \cup E(G) \to V(H) \cup E(H)$ such that if u, v are vertices of G and e is an edge such that $\{u, v\} = \text{Ends}(e)$ then $\{\gamma(u), \gamma(v)\} = \text{Ends}(\gamma(e))$. Since we deal only with graphs without self-loop, we have $\gamma(u) \neq \gamma(v)$ whenever $\{u, v\}$ is an edge of G. Note also that $\gamma(I_G(u)) \subseteq I_H(\gamma(u))$.

We say that the homomorphism γ is an isomorphism if γ is bijective. We write $G \simeq G'$ whenever G and G' are isomorphic. A class of graphs will be any set of graphs containing all graphs isomorphic to some of its elements. The class of all graphs will be denoted by $\mathcal{G}$. Given a graph G, a graph H whose vertices and edges are all in G is a subgraph of G. For an edge e of G, we denote by $A_G(e)$ the single edge subgraph $(\text{Ends}(e), \{e\})$. An occurrence of G in G' is an isomorphism γ between G and a subgraph H of G'.

For any set S, $|S|$ denotes the cardinality of S. For any integer q, we denote by $[1, q]$ the set of integers $\{1, 2, \ldots, q\}$.

2.2 Labelled Graphs

Throughout the paper we will consider graphs where vertices and edges are labelled with labels from a recursive set L. A graph labelled over L will be denoted by (G, λ), where G is a graph and $\lambda\colon V(G) \cup E(G) \to L$ is the labelling function. The graph G is called the underlying graph and the mapping λ is a labelling of G. The class of labelled graphs over some fixed alphabet L will be denoted by $\mathcal{G}_L$. Note that since L is recursive, $\mathcal{G}_L$ is also recursive.

Let (G, λ) and (G', λ') be two labelled graphs. Then (G, λ) is a subgraph of (G', λ'), denoted by $(G, \lambda) \subseteq (G', \lambda')$, if G is a subgraph of G' and λ is the restriction of the labelling λ' to $V(G) \cup E(G)$.

A mapping $\gamma\colon V(G) \to V(G')$ is a homomorphism from (G, λ) to (G', λ') if γ is a graph homomorphism from G to G' which preserves the labelling, i.e., such that $\lambda'(\gamma(x)) = \lambda(x)$ holds for every $x \in V(G) \cup E(G)$.

An *occurrence* of (G, λ) in (G', λ') is an isomorphism γ between (G, λ) and a subgraph (H, η) of (G', λ'). It shall be denoted by $\gamma : (G, \lambda) \hookrightarrow (G', \lambda')$.

Labelled graphs will be designated by bold letters like $\mathbf{G}$, $\mathbf{H}$, ... If $\mathbf{G}$ is a labelled graph, then G denotes the underlying graph.

Remark 3 The labelling λ of vertices may encode some properties of the network or an initial knowledge. For example, if the network is anonymous, all the vertices have the same label (i.e., $\forall u, u' \in V(G), \lambda(u) = \lambda(u')$). If the processes have unique identities, then for all $u, u' \in V(G)$ if $u \neq u'$ then $\lambda(u) \neq \lambda(u')$. If there exists a distinguished process, then there exists $u \in V(G)$ such that for each $u' \in V(G)$ distinct from u, $\lambda(u) \neq \lambda(u')$. It may also encode partial identities of processes. Same properties for edges may be encoded by λ. For example, labels of edges may encode a spanning-tree. The degree of a vertex or the size of the graph are examples of initial knowledge.

3 Local computations on (open) labelled edges and distributed algorithms

In this section we give definitions of local computations on (open) labelled edges, their interpretation as distributed algorithms and an example of distributed algorithms encoded by such local computations. Our terminology is inspired by definitions of [3].

3.1 Local computations on labelled edges

Let us recall that informally local computations on labelled edges are applied to labelled graphs and they modify, at each step, the labels of an edge and the labels of its endvertices according to a rule depending on the labels of the edge and its endvertices.

Let $\mathcal{R}$ be a set of relabelling rules as defined in Fig. 1 of Introduction. Let $\mathbf{G}_1$ and $\mathbf{G}_2$ be two labelled graphs. We say that $\mathbf{G}_1$ yields $\mathbf{G}_2$ in one step, written $\mathbf{G}_1 \ \mathcal{R} \ \mathbf{G}_2$, if and only if there exists an edge $e = \{v_1, v_2\}$ of $\mathbf{G}_1$, a rule $R = ((X, Y, Z), (X', Y', Z'))$ of $\mathcal{R}$ such that:

- in $\mathbf{G}_1$, X is the label of v_1, Y is the label of e, and Z is the label of v_2;
- $\mathbf{G}_2$ is obtained from $\mathbf{G}_1$ by replacing the label X of v_1 by X', the label Y of e by Y', and the label Z of v_2 by Z'.

As usual, $\mathcal{R}^*$ denotes the reflexive and transitive closure of $\mathcal{R}$. Local computations on labelled edges on graphs are computations on graphs corresponding to relabelling relations obtained by this way; they are also called locally generated relabelling relations on labelled edges.

A sequence $(\mathbf{G}_i)_{0 \leq i \leq n}$ is called a $\mathcal{R}$-*relabelling sequence* (or *relabelling sequence,* when $\mathcal{R}$ is clear from the context) if $\mathbf{G}_i \ \mathcal{R} \ \mathbf{G}_{i+1}$ for every $0 \leq i < n$ (with n being the length of the sequence). A relabelling sequence of length 1 is a *relabelling step.*

Let $\mathcal{R}$ be a relabelling system; by definition $\mathcal{R}$ has the termination property if there is no infinite $\mathcal{R}$-relabelling sequence.

Notation Let $\mathcal{R}$ be a relabelling relation. Let $\mathbf{G}$ and $\mathbf{G}'$ be two labelled graphs; $\mathbf{G} \ \mathcal{R} \ \mathbf{G}'$ will be denoted also by $\mathbf{G} \underset{\mathcal{R}}{\Longrightarrow} \mathbf{G}'$.

A complete formal presentation of all notions attached to graph relabelling systems may be found in [16] (Sect. 3).

Remark 4 We only consider recursive set of relabelling rules such that the set of irreducible graphs is recursive. The purpose of all assumptions about recursiveness done throughout the paper is to have "reasonable" objects w.r.t. their computational power.

3.2 Local computations on open labelled edges

In this paper, we consider also the following model of computation: at each step of computation labels are modified on exactly one edge and one endvertex of this edge of the given graph, according to certain rules depending on the label of this edge and the labels of its endvertices only; it is presented in Fig. 2. As in the previous subsection, we define graph relabelling systems on open labelled edges and locally generated relabelling relations on open labelled edges. Such local computations are called local computations on open labelled edges.

3.3 Distributed algorithms

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

Hence, our notion of relabelling sequence associated to a locally generated relabelling relation may be regarded as a *serialization* [19] of a distributed computation. This model is asynchronous, in the sense that several relabelling steps *may* be done at the same time but we do not require that all of them have to be performed. In the sequel we will essentially handle

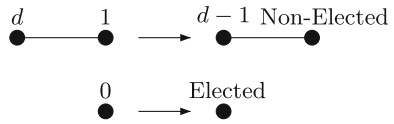


Fig. 5 The rewriting system ELECTION- TREE

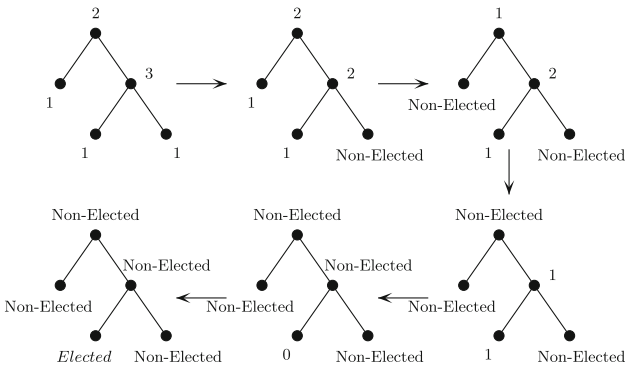


Fig. 6 An example of a run of ELECTION- TREE

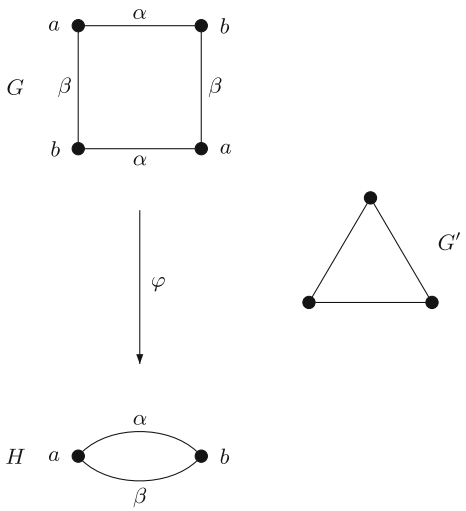


Fig. 7 The graphs G and G' are minimal if we only consider simple graphs. The graph G' is minimal while the graph G is not minimal if we consider graphs with multiple edges

sequential relabelling sequences, but the reader should keep in mind that such sequences may be done in parallel.

Finally in the model of local computations, $\mathcal{R}$ is regarded as a distributed algorithm and we will denote this algorithm by $\mathcal{R}$. We give below an example of distributed algorithm encoded by local computations that solves leader election on trees when each vertex initially knows its degree.

Example 5 The rewriting system described in Fig. 5, denoted ELECTION- TREE, elects in any tree such that initially each vertex is labelled by its degree. An execution of this algorithm is presented in Fig. 6.

4 Election, enumeration and local computations on labelled edges

In this section, we define the problem of election, that consists in distinguishing a vertex from the others, and the problem of enumeration, that consists in giving a unique number to each one of the vertices of the graph.

We present coverings and we recall an impossibility result about the election problem (and the enumeration problem).

4.1 Definitions

A distributed election algorithm on a labelled graph **G** is a graph relabelling system on labelled edges such that the result of any computation is a labelling of the vertices verifying: exactly one vertex has the label *elected* and all other vertices have the label *non-elected*. The labels *elected* and *non-elected* are terminal, i.e., when they appear on a vertex they remain until the end of the computation.

A distributed enumeration algorithm on a labelled graph **G** is a graph relabelling system on labelled edges such that the result of any computation is a labelling of the vertices that is a bijection from $V(G)$ to $\{1, 2, \ldots, |V(G)|\}$. It is easy to see that if we have an enumeration algorithm on a graph **G** where vertices can detect whether the algorithm has terminated, we have an election algorithm on **G** by electing the vertex labelled by 1.

4.2 Coverings

We say that a graph G is a *covering* of a graph H via γ if γ is a surjective homomorphism from G onto H such that for every vertex v of $V(G)$ the restriction of γ to $I_G(v)$ is a bijection onto $I_H(\gamma(v))$. The covering is proper if G and H are not isomorphic.

The notion of covering extends to labelled graphs in an obvious way. The labelled graph (H, λ') is *covered* by (G, λ) via γ, if G is a *covering* of graph H via γ and γ is label-preserving.

Remark 6 We use a different definition for coverings than Angluin's one. In fact, if we consider only simple graphs these two definitions are equivalent. For Angluin, (H, λ') is *covered* by (G, λ) via γ, if γ is a homomorphism from (G, λ) to (H, λ') such that for every vertex v of $V(G)$ the restriction of γ to $N_G(v)$ is a bijection onto $N_H(\gamma(v))$. Given a simple graph G, for each vertex $v \in V(G)$, there is a natural bijection between $I_G(v)$ and $N_G(v)$ and therefore it is easy to see the equivalence.

We work with graphs that can have multiple edges and in this case the two definitions are not equivalent. Consider the graphs G and H from Fig. 7, if we consider the morphism

φ defined from G to H by the letters a, b, α, β, we easily see that G is a covering of H. But if we use Angluin's definition of covering, G is not a covering of H since for each $u \in G$, $|N_G(u)| = 2$, whereas for each $v \in H$, $|N_H(v)| = 1$.

A labelled graph $\mathbf{G}$ is called *minimal* if every covering from $\mathbf{G}$ to some $\mathbf{H}$ is a bijection. Complete graphs are examples of minimal graphs. The graphs G' and H from Fig. 7 are minimal graphs, whereas G is a proper covering of H and therefore G is not minimal. Moreover, G is not a proper covering of any simple graph.

Remark 7 From a complexity point of view it has been shown [10] that deciding if a given graph G is minimal is co-NP-complete.

We have the following basic property of coverings (see [24]):

Lemma 8 ([24]) *For every covering γ from $\mathbf{G}$ to $\mathbf{H}$ there exists an integer q such that $|\gamma^{-1}(x)| = q$, for all $x \in V(H) \cup E(H)$.*

The integer q in the previous lemma is called the number of *sheets* of the covering. We also refer to γ as a *q-sheeted covering*.

From Lemma 8, we deduce that for any graph $\mathbf{G}$ such that $|V(G)|$ and $|E(G)|$ are coprimes, $\mathbf{G}$ is minimal. Examples of such graphs are trees, k-trees, graphs with a prime number of vertices (or edges). If the number of distinct labels appearing on the vertices of $\mathbf{G}$ is coprime with the size of G, again, we can deduce that $\mathbf{G}$ is minimal.

The following lemma shows that two edges that have the same image by a covering cannot have a common end vertex.

Lemma 9 *Let $\mathbf{G}$ be a covering of $\mathbf{H}$ via γ and let $e_1, e_2 \in E(G)$ be such that $e_1 \neq e_2$. If $\gamma(e_1) = \gamma(e_2)$ then $A_G(e_1) \cap A_G(e_2) = \emptyset$, i.e., $Ends(e_1) \cap Ends(e_2) = \emptyset$.*

4.3 Local computations and coverings

We now present the fundamental lemma connecting coverings and locally generated relabelling relations on labelled edges due to Angluin [3]. It states that, whenever $\mathbf{G}$ is a covering of $\mathbf{H}$, every relabelling step in $\mathbf{H}$ can be lifted to a relabelling sequence in $\mathbf{G}$, which is compatible with the covering relation.

Lemma 10 (Lifting Lemma [3]) *Let $\mathcal{R}$ be a locally generated relabelling relation on labelled edges and let $\mathbf{G}$ be a covering of $\mathbf{H}$ via γ. If $\mathbf{H}\ \mathcal{R}^*\ \mathbf{H}'$ then there exists $\mathbf{G}'$ such that $\mathbf{G}\ \mathcal{R}^*\ \mathbf{G}'$ and $\mathbf{G}'$ is a covering of $\mathbf{H}'$ via γ.*

Proof It suffices to show the claim for the case $\mathbf{H}\ \mathcal{R}\ \mathbf{H}'$. Suppose that the relabelling step changes labels in $A_H(e)$, for some edge $e \in E(H)$. We may apply this relabelling step to each of the disjoint labelled single edge graphs of $\gamma^{-1}(A_H(e))$, since they are isomorphic to $A_H(e)$. This yields $\mathbf{G}'$ which satisfies the claim.

This is depicted in the following commutative diagram:

$$\begin{array}{ccc} \mathbf{G} & \xrightarrow[\mathcal{R}^*]{} & \mathbf{G}' \\ {\scriptstyle\gamma}\downarrow & & \downarrow{\scriptstyle\gamma} \\ \mathbf{H} & \xrightarrow[\mathcal{R}^*]{} & \mathbf{H}' \end{array}$$

4.4 Impossibility result

We prove that there exists no enumeration algorithm and no election algorithm on a graph $\mathbf{G}$ if the graph is not minimal. To show this result, we use the same method as in the Lifting Lemma [3]. If we assume there exists an enumeration (or an election) algorithm we deduce that there exists a run which either does not terminate or it terminates and there are two vertices having the same number (or two elected vertices). In any case we obtain a contradiction, and:

Proposition 11 *Let $\mathbf{G}$ be a labelled graph which is not minimal, there is no enumeration (election) algorithm for $\mathbf{G}$.*

5 An enumeration algorithm for minimal graphs based on local computations on labelled edges

In this section, we describe an algorithm $\mathcal{M}$ using local computations on labelled edges that solves the enumeration problem on a minimal labelled graph $\mathbf{G}$. This algorithm is related and uses some ideas developed in [20]. Each vertex v attempts to get its own number between 1 and $|V(G)|$. A vertex chooses a number and broadcasts it with its label and its labelled neighbourhood all over the network. If a vertex u discovers the existence of another vertex v with the same number, then it compares its *local view*, i.e., the labels and numbers of its neighbours, with the local view of v. If the label of u or the local view of u is "weaker" (for an order defined below), then u chooses another number and broadcasts it again with its local view. At the end of the computation, every vertex will have a unique number if the graph is minimal.

5.1 Labels

5.1.1 Labels for edges

For each edge $e \in E(G)$ a number $p(e)$ will be associated to e such that at the end of the run for each endvertex v of e if $e, e' \in I_G(v)$ and $e \neq e'$ then $p(e) \neq p(e')$. The label of

an edge e is the pair $(\lambda(e), p(e))$ and the initial labelling is $(\lambda(e), 0)$.

Remark 12 The label $p(e)$ for an edge e simulates a port numbering.

5.1.2 Labels for vertices

For each vertex $v \in V(G)$, the label of v is of the form $(\lambda(v), n(v), N(v), M(v))$ where:

- $n(v) \in \mathbb{N}$ is the *number* of the vertex v computed by the algorithm;
- $N(v) \in \mathcal{P}_{\text{fin}}(\mathbb{N} \times L \times \mathbb{N})$[2] is the *local view* of v, and it is a tuple defined by:

$$N(v) = \{(p(e), \lambda(e), n(v')) \mid e \in I_G(v), \\ \text{Ends}(e) = \{v, v'\} \text{ and } p(e) \neq 0\};$$

- $M(v) \subseteq \mathbb{N} \times L \times \mathcal{P}_{\text{fin}}(\mathbb{N} \times L \times \mathbb{N})$ is the *mailbox* of v and contains the whole information received by v at any step of the computation; for each element (n_0, ℓ_0, N_0) of $M(v)$ there exists a vertex u and a step i such that at step i, $n(u) = n_0$, $N(u) = N_0$ and $\lambda(u) = \ell_0$.

The initial labelling of any vertex v is $(\lambda(v), 0, \emptyset, \emptyset)$.

5.2 An order on local views

The fundamental property of the algorithm is based on a total order on local views, as defined in [20], such that the local view of any vertex cannot decrease during the computation. We assume for the rest of this paper that the set of labels L is totally ordered by $<_L$. We consider the lexicographic order on $\mathbb{N} \times L \times \mathbb{N}$, i.e., $(p, \ell, n) \prec (p', \ell', n')$ if $p < p'$ or $(p = p'$ and $\ell <\ell')$ or $(p = p'$ and $\ell = \ell'$ and $n < n')$. Then we define the order $\prec$ on $\mathcal{P}_{\text{fin}}(\mathbb{N} \times L \times \mathbb{N})$ by: $N_1 \prec N_2$ if the maximum of the symmetric difference $N_1 \triangle N_2 = (N_1 \setminus N_2) \cup (N_2 \setminus N_1)$ belongs to N_2.

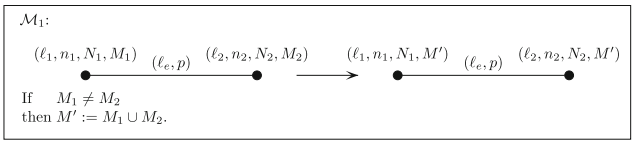
If $N(u) \prec N(v)$, then we say that the local view $N(v)$ of v is stronger than the one of u and that $N(u)$ is weaker than $N(v)$. Using the total order $<_L$ over L, the order $\prec$ is extended to an order over $L \times \mathcal{P}_{\text{fin}}(\mathbb{N} \times L \times \mathbb{N})$: $(\ell, N) \prec (\ell', N')$ if $\ell <_L \ell'$ or if $\ell = \ell'$ and $N \prec N'$. In the sequel, the reflexive closure of $\prec$ will be denoted by $\preceq$.

5.3 Relabeling rules

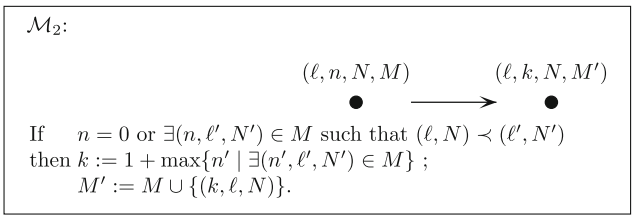
We describe here the relabelling rules that define the enumeration algorithm.

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

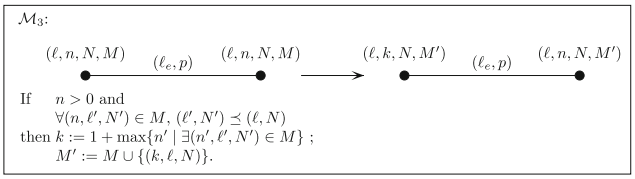
The first rule $\mathcal{M}_1$ enables two adjacent vertices to update their mailboxes if they are not equal.



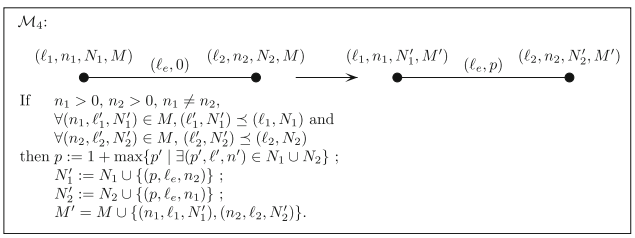
The second rule $\mathcal{M}_2$ does not involve any synchronization with a neighbour vertex. It enables a vertex v to change its identity if the current identity number $n(v)$ is 0 or if the mailbox of v contains a message from a vertex with the same identity but with a stronger label or a stronger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



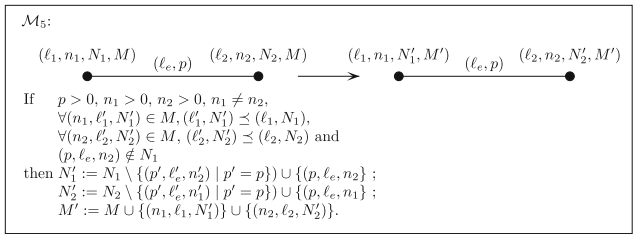
The third rule $\mathcal{M}_3$ allows to change the current identity for a vertex v having a neighbour v' with exactly the same current label (all four components should be identical). This relabelling step can be applied only if the rule $\mathcal{M}_2$ cannot be applied by v or v'.



The fourth rule is applied on an edge $e = \{v, v'\}$ if e has not yet a number, i.e., $p(e) = 0$. We will prove that the choosen number is such that it is different from numbers of the other incident edges to v or v'. Then we update local views and mailboxes of v and v'. This rule is applied on the edge e if rules $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$ cannot be applied on e, v or v'.



The fifth rule is applied on an edge $e = \{v, v'\}$ if it is necessary to update local views of v or v', i.e., $(p(e), \lambda(e), n(v')) \notin N(v)$ or $(p(e), \lambda(e), n(v)) \notin N(v')$. This rule cannot be applied if one of the preceeding rules can be applied.



5.4 Main properties and proof of $\mathcal{M}$

Let $\mathbf{G}$ be a simple labelled graph. In the following i is an integer denoting a computation step. Let $(\lambda(v), n_i(v), N_i(v), M_i(v))$ be the label of the vertex v after the ith step of the computation and let $(\lambda(e), p_i(e))$ be the label of the edge e after the ith step.

A careful examination of the relabelling rules shows:

Lemma 13 *For all edges $e, e' \in E(G)$, for any vertex $v \in V(G)$, and at each step i,*

1. $p_i(e) \neq 0 \implies p_{i+1}(e) = p_i(e)$,
2. $\exists (p, \ell_e, n) \in N_i(v) \iff \exists e \in I_G(v)$ *such that* $p_i(e) = p > 0$ *and* $\lambda(e) = \ell_e$,
3. $(p_i(e) \neq 0$ *and* $p_i(e') \neq 0$ *and* $Ends(e) \cap End(e') \neq \emptyset)$ $\implies p_i(e) \neq p_i(e')$,
4. $n_i(v) \neq 0 \implies (n_i(v), \lambda(v), N_i(v)) \in M_i(v)$,
5. $\forall (p, \ell_e, n) \in N_i(v) : n \neq 0$, $p \neq 0$ *and* $\exists (n, \ell', N') \in M_i(v)$,
6. $\nexists (p, \ell_e, n_i(v)) \in N_i(v)$,
7. $\forall (p, \ell_e, n), (p', \ell_e', n') \in N_i(v)$, $p \neq p'$.

We now prove an increasing lemma that enables to prove the termination of the algorithm.

Lemma 14 *For each vertex v and each step i:*

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

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

For each vertex v and each step i such that $n_i(v) \neq n_{i+1}(v)$, $n_{i+1}(v) = 1 + \max\{n_1; (n_1, \ell_1, N_1) \in M_i(v)\}$ and either $n_i(v) = 0 < n_{i+1}(v)$ or $(n_i(v), \lambda(v), N_i(v)) \in M_i(v)$ as shown in Lemma 13 and therefore $n_i(v) < n_{i+1}(v)$.

For each vertex v such that $N_i(v) \neq N_{i+1}(v)$, either a new element has been added to $N_i(v)$ and therefore $N_i(v) \preceq N_{i+1}(v)$, or $N_{i+1}(v) = N_i(v) \setminus \{(p, \ell_e, m)\} \cup \{(p, \ell_e, m')\}$ where $m' = n_{i+1}(v')$ whereas $m = n_{i_0}(v')$ for a step $i_0 \leq i$; therefore $m' > m$ and consequently, $N_i(v) \preceq N_{i+1}(v)$.

Furthermore, one of the inequalities is strict for at least one vertex, since each time a rule is applied, there exists at least one vertex whose label is modified.

Lemma 15 *For every $v \in V(G)$ and for every step i if $(m, \ell, N) \in M_i(v)$ then there exists a vertex $w \in V(G)$ such that $n_i(w) = m$.*

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

Let v be a vertex, let i be a step of computation and let (m, ℓ, N) be an element of $M_i(v)$; we denote by U the set $\{(u, j) \in V(G) \times \mathbb{N} \mid j \leq i, n_j(u) = m\}$.

Let U' be the set defined by $U' = \{(u, j) \in U \mid \forall (u', j') \in U, (\lambda(u'), N_{j'}(u')) \prec (\lambda(u), N_j(u))$ or $(\lambda(u'), N_{j'}(u')) = (\lambda(u), N_j(u))$ and $j' \leq j\}$.

By hypothesis, $(m, \ell, N) \in M_i(v)$, thus U et U' are two nonempty sets. Furthermore,there exists i_0 such that for all $(u, j) \in U'$, $j = i_0$.

If $i_0 < i$ then there exists exactly one element $(u, i_0) \in U'$ (at each step the number of at most one vertex is modified). Thus the number $n_{i_0}(u) = m$ has been modified at step $i_0 + 1$. This is not possible because at this step the vertex u had no neighbour with the same name m, thus the rule $\mathcal{M}_3$ has not been applied, and as $(\lambda(u), N_{i_0}(u))$ is maximal the rule $\mathcal{M}_2$ has not been applied on u at the step i_0. Finally, $i_0 = i$ and there exists a vertex w such that $n_i(w) = m$.

Lemma 16 *For every vertex $v \in V(G)$ and every step i such that $n_i(v) \neq 0$, given $(m', \ell', N') \in M_i(v)$, for every $1 \leq m \leq m'$, there exists $(m, \ell, N) \in M_i(v)$.*

Proof We show this claim by induction on i. For $i = 0$ the property is true. We assume that the property holds for $i \geq 0$. This property remains true at step $i + 1$ for every vertex $w \in V(G)$ not modified by this step. Let v be a vertex whose label is modified at step $i + 1$.

If the rule $\mathcal{M}_1$ is applied at step $i + 1$ on v and on a neighbour of v, then $M_{i+1}(v) = M_i(v) \cup M_i(v')$ and the property is still verified at step $i + 1$ because it is true for v and v' at step i.

If the rule $\mathcal{M}_2$ or the rule $\mathcal{M}_3$ is applied on v at step $i + 1$, then $M_{i+1}(v) = M_i(v) \cup \{1 + \max\{m \mid (m, \ell, N) \in M_i(v)\}, \lambda(v), N_i(v))\}$, finally for each $m \in M_{i+1}(v)$, the property remains true.

If the rule $\mathcal{M}_4$ or the rule $\mathcal{M}_5$ is applied on v at step $i + 1$, for all $(m, \ell, N) \in M_{i+1}(v)$, there exists $(m, \ell, N') \in M_i(v)$ and the property remains true.

Lemma 17 *Any run ρ of the enumeration algorithm on a connected labelled graph $\mathbf{G} = (G, \lambda)$ terminates.*

Proof From Lemmas 15 and 16, the numbers that can have the different vertices are between 0 and $|V(G)|$ and once an edge get a number, its label never changes. Consequently, $N(v)$ and $M(v)$ can also take only a finite number of values and therefore, it follows from Lemma 14 that the algorithm terminates.

If v is a vertex of G and e is an edge of G then the label of v after a run ρ of the enumeration algorithm is denoted by $(\lambda(v), c_\rho(v))$ with $c_\rho(v) = (n_\rho(v), N_\rho(v), M_\rho(v))$ and the label of e is denoted by $(\lambda(e), p_\rho(e))$.

Lemma 18 *Any run ρ of the enumeration algorithm on a connected labelled graph $\mathbf{G} = (G, \lambda)$ terminates and yields a final labelling (λ, c_ρ) verifying the following conditions for all vertices v, v' of G:*

1. *Let m be the maximal number in the final labelling, $m = \max_{v \in V(G)} n_\rho(v)$. Then for every $1 \leq k \leq m$, there is some $v \in V(G)$ with $n_\rho(v) = k$.*
2. *$M_\rho(v) = M_\rho(v')$.*
3. *$(n_\rho(v), \lambda(v), N_\rho(v)) \in M_\rho(v')$.*
4. *If $n_\rho(v) = n_\rho(v')$ then $(\lambda(v) = \lambda(v')$ and $N_\rho(v) = N_\rho(v'))$.*
5. *for any edges $e, e' \in I_G(v)$, $p_\rho(e) > 0$, $p_\rho(e') > 0$ and $p_\rho(e) \neq p_\rho(e')$.*
6. *$(p, \ell, n) \in N_\rho(v)$ if and only if there exists an edge e incident to v such that $p_\rho(e) = p$ and $\lambda(e) = \ell$. Furthermore, if $Ends\{e\} = \{v, w\}$, then $n_\rho(w) = n$ and $(p, \ell, n_\rho(v)) \in N_\rho(w)$.*

Proof 1. By Lemmas 15 and 16 applied to the final labelling.
2. Otherwise, the rule $\mathcal{M}_1$ could be applied.
3. A direct corollary of the previous property using Lemma 13.
4. Otherwise the rule $\mathcal{M}_2$ could be applied to v or v'.
5. Otherwise, the rule $\mathcal{M}_4$ could be applied and they are different by Lemma 13.
6. From Lemma 13 and otherwise $\mathcal{M}_3$, $\mathcal{M}_4$ or $\mathcal{M}_5$ could be applied.

Proposition 19 *Given a graph labelled $\mathbf{G}$ and an execution ρ the enumeration algorithm on $\mathbf{G}$, a labelled graph $\mathbf{H}$ is associated to the final labelling such that $\mathbf{G}$ is a covering of $\mathbf{H}$.*

Proof Let m be the maximal number in the final labelling and consider the graph H such that $V(H) = [1, m]$ and $E(H) = \{(p, \{n_\rho(v), n_\rho(v')\}) \mid \exists e \in E(G),\ \text{Ends}(e) = \{v, v'\},\ p_\rho(e) = p\}$ where for each $(p, \{n, n'\}) \in E(H)$, $\text{Ends}((p, \{n, n'\})) = \{n, n'\})$.

Consider the homomorphism $\varphi : G \to H$ such that for each $v \in V(G)$, $\varphi(v) = n_\rho(v)$ and for each $e \in E(G)$ such that $\text{Ends}(e) = \{v, v'\}$, $\varphi(e) = (p_\rho(e), \{n_\rho(v), n_\rho(v')\})$. For every $v, v' \in V(G)$, if $n(v) = n(v')$ then $\lambda(v) = \lambda(v')$, we can therefore define $\lambda(\varphi(v)) = \lambda(v)$. Moreover, if an edge e is such that $\text{Ends}(e) = \{v, v'\}$, it means that $(p_\rho(e), \lambda(e), n_\rho(v'), \lambda(v')) \in N(v)$ and $(p_\rho(e), \lambda(e), n_\rho(v), \lambda(v)) \in N(v')$: we can therefore define $\lambda(\varphi(e)) = \lambda(e)$.

For every vertex $v \in V(G)$ and for every edges $e, e' \in I_G(v)$, if $e \neq e'$, we know that $p_\rho(e) \neq p_\rho(e')$ and consequently, $\varphi(e) \neq \varphi(e')$: φ is an injection from $I_G(v)$ into $I_H(\varphi(v))$.

For every vertex $v \in V(G)$, for every edge $f \in I_H(\varphi(v))$, there exists $v', v'' \in V(G)$ and $e \in E(G)$ such that $\text{Ends}(e) = \{v', v''\}$ and $\varphi(e) = f$. Since $\text{Ends}(f) = \{\varphi(v'), \varphi(v'')\}$, we can assume that $\varphi(v') = \varphi(v)$ and therefore $N(v) = N(v')$ and $(p_\rho(e), \lambda(e), n_\rho(v''), \lambda(v'')) \in N(v)$. Consequently, there exists $e' \in I_G(v)$ such that $\varphi(e) = \varphi(e')$ and therefore φ is a surjection from $I_G(v)$ into $I_H(\varphi(v))$.

Consequently, φ is a surjective homomorphism from $\mathbf{G}$ into $\mathbf{H}$ such that for each vertex $v \in V(G)$, its restriction to $I_G(v)$ is a bijection to $I_H(\phi(v))$: $\mathbf{G}$ is a covering of $\mathbf{H}$ via φ.

Given a minimal graph $\mathbf{G}$, if $\mathbf{G}$ is a covering of a graph $\mathbf{H}$, $\mathbf{G} \simeq \mathbf{H}$. Consequently, for every run ρ of the enumeration algorithm, the graph associated to the final labelling is isomorphic to $\mathbf{G}$ and the set of numbers the vertices have is exactly $\{1, \ldots, |V(G)|\}$. Moreover, it has been shown in Lemma 11 that for every graph $\mathbf{G}$ that is not minimal, there exists no algorithm to solve the enumeration problem on $\mathbf{G}$.

The termination detection of the algorithm is possible on a minimal graph $\mathbf{G}$. Indeed, once a vertex gets the number $|V(G)|$, from Lemmas 15 and 16, it knows that all the vertices have a different number and therefore it can detect the termination. Finally:

Theorem 20 *For every labelled graph $\mathbf{G}$, the following properties are equivalent:*

1. *there exists an enumeration algorithm for $\mathbf{G}$ based on local computations on labelled edges,*
2. *there exists an election algorithm and an enumeration algorithm with termination detection for $\mathbf{G}$ based on local computation on labelled edges,*
3. *$\mathbf{G}$ is minimal for the covering relation.*

Remark 21 Let $\mathbf{G}$ be a minimal graph for the covering relation, a vertex can detect that each vertex has a unique name if it knows the size of the graph (it does not need to know exactly $\mathbf{G}$). It means that for each n, there exists a unique election (or naming) algorithm for the class of minimal grpahs of size n.

5.5 Analysis of $\mathcal{M}$

In this subsection, we study two aspects of the complexity of $\mathcal{M}$: the maximal length of sequences of any run for a given graph and the maximum local memory requirement at any vertex. First we give an upper bound for the length of sequences for any run of $\mathcal{M}$ on a graph $\mathbf{G}$ of size n and maximal vertex degree Δ. We show that any execution of our algorithm takes a polynomial number of steps, and that when executing our algorithm, each process needs a memory of polynomial size (in the size of the graph).

Proposition 22 *Let* $\mathbf{G}$ *be a labelled graph of size* n *and maximal vertex degree* Δ. *For any run of* $\mathcal{M}$ *on* $\mathbf{G}$ $O(\Delta n^3)$ *rules are applied.*

Proof Let $\mathbf{G}$ a labelled graph of size n and a run ρ on $\mathcal{M}$ n $\mathbf{G}$. From Lemmas 15 and 16, we deduce that rules $\mathcal{M}_2$ et $\mathcal{M}_3$ cannot be applied more that $\frac{n(n-1)}{2}$ times. Between two steps where rules $\mathcal{M}_2$ or $\mathcal{M}_3$ are applied, rules $\mathcal{M}_4$ and $\mathcal{M}_5$ are applied at most Δ times (once for each neighbour of the relabelled vertex). Thus rules $\mathcal{M}_4$ and $\mathcal{M}_5$ are applied $O(\Delta n^2)$ times.

Each time a vertex v modifies its number or its local view, a triple of the form (n_0, ℓ, N) is put in $M(v)$. For each triple, the rule $\mathcal{M}_1$ is applied at most n times.

The result follows.

Now we study the maximal local memory requirement, i.e., the size of the label added to a vertex. We assume that λ (the labelling of $\mathbf{G}$) is such that the initial label on any vertex ℓ needs $O(\log |V(G)|)$ memory bits.

Remark 23 We can note that the mailbox of each vertex contains a lot of useless information. Indeed, if some (m, ℓ, N) belongs to the mailbox $M(v)$ of a vertex v, one can remove from $M(v)$ all the elements $(m, \ell', N') \in M(v)$ such that $(\ell', N') \prec (\ell, N)$. We can thus replace the mailbox $M(v)$ of v by $\{(m, \ell, N) \in M(v) \mid \forall (m, \ell', N') \in M(v), (\ell', N') \preceq (\ell, N)\}$. In this way, the mailbox of each vertex contains at most $|V(G)|$ elements of the form (m, ℓ, N). In the sequel we assume that the mailboxes verify this property.

Proposition 24 *Let* $\mathbf{G}$ *be a labelled graph of size* n *and maximal degree vertex* Δ. *Any run of* $\mathcal{M}$ *needs* $O(\Delta n \log n)$ *memory bits on each vertex.*

Proof Let $\mathbf{G}$ be a labelled graph of size n with m edges and maximal vertex degree Δ. The label of each edge is modified exactly once: it is a number upper bounded by m. The local view of any vertex is at most Δ triples (p, ℓ_e, n_0) of $O(\log n)$ memory bits. Thus for any vertex v, the memory requirement for $N(v)$ is $O(\Delta \log n)$.

Each vertex needs in its mailbox the following set $\{(n_0, \ell, N) \in M(v) \mid \forall (n_0, \ell', N') \in M(v), (\ell', N') \preceq (\ell, N)\}$. Thus, in the mailbox of each vertex, there is at most n triples (n_0, ℓ, N) and the memory requirement is $O(\Delta \log n)$ memory bits. Finally, the memory requirement for the mailbox of each vertex is $O(\Delta n \log n)$ memory bits.

Remark 25 The algorithm of Mazurkiewicz [20] is also an algorithm that is executed in a polynomial number of steps, and where each process needs a memory of polynomial size [17].

On the contrary, the algorithms of Yamashita and Kameda [28], and of Boldi et al. [5] run in a linear number of rounds, but each process needs an exponential memory.

6 Application to Angluin's model

6.1 Angluin's model

The model of Angluin [3] is defined in the following way. The communication model is a point-to-point communication network which is represented as a simple connected undirected graph where vertices represent processes and two vertices are linked by an edge if the corresponding processes have a direct communication link. Processes communicate by message passing and each process can distinguish its neighbours, i.e., the different links incident to it. Since each process knows from which channel it receives a message or it sends a message, one suppose that the network is represented by a simple graph with a port numbering function (see Fig. 8).

Definition 26 Given a simple labelled graph $\mathbf{G}$, a *port numbering* function δ is a set of local functions $\{\delta_u \mid u \in V(G)\}$ such that for each vertex $u \in V(G)$, δ_u is a bijection between $I_G(u)$ and $[1, \deg_G(u)]$.

There is no global time (the network is asynchronous). A basic computation step is a pairwise exchange of messages by the two processes at the two ends of some edge. This exchange allows the two processes to change their internal states and does not affect any other process.

To break the symmetry between two adjacent processes a non-deterministic "coin toss" may be used. Let A be the set of messages, the behaviour of a process of degree d is

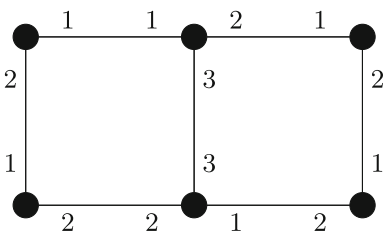


Fig. 8 A graph G with a port numbering

Fig. 9 Graphical form of a rule of Angluin's model, where $X' = f(X, Z, i, 0)$, $Z' = f(Z, X, j, 1)$, i and j are the corresponding port numbers and f is the transition function

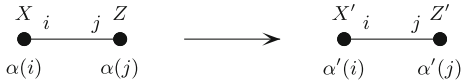


Fig. 10 A relabelling rule on an edge with port-numbering

Fig. 11 A relabelling rule on a labelled edge

defined by a pair (Q, M) such that Q is nonempty (possibly infinite) set of states and M is a map called a communication function from $Q \times [1, d]$ to subsets of $A \times A \times Q$.

Asynchrony is modelled by considering sets of sequences corresponding to different interleavings of the basic steps.

Thus a basic computation step has the form given in Fig. 9.

The port numbers of the edges incident to a node may be used by the process located at that node to memorize the labels (states) associated to the corresponding incident edges. This fact implies that relabelling on edges with port numbering can simulate local computations on labelled edges, i.e., rules having the form given in Fig. 10 simulates rules having the form given in Fig. 11 with $\alpha(i) = \alpha(j) = Y$ and $\alpha'(i) = \alpha'(j) = Y'$.

Finally:

Lemma 27 *Local computations as they are defined in Angluin's model can simulate local computations on labelled edges.*

6.2 Election in Angluin's model

Given a labelled graph **G**, as in [29], we ask in Angluin's model that there exists an election algorithm for every port numbering δ of **G**.

First, the notion of port numbering may be extended to graphs having multiple edges.

Definition 28 Let **G** be a labelled graph having possibly multiple edges. A port numbering δ is a set of functions $\{\delta_u \mid u \in V(G)\}$ such that for any vertex u, δ_u is a bijection between $I_G(u)$ and $[1, \deg_G(u)]$.

The definition of coverings can be extended in a natural way to graphs with a port numbering:

Definition 29 A labelled graph **G** with a port numbering δ is a covering of a labelled graph **H** with a port numbering δ' via γ if **G** is a covering of **H** via γ and if for any edge $e \in E(G)$ and for any vertex $u \in Ends\{e\}$, $\delta_u(e) = \delta_{\gamma(u)}(\gamma(e))$.

Lemma 30 *For any labelled graphs* **G** *and* **H** *such that* **G** *is a covering of* **H** *via* γ*, for any port numbering* δ' *of* **H** *there exists a port numbering* δ *such that* $(\mathbf{G}, \delta)$ *is a covering of* $(\mathbf{H}, \delta')$.

Proof It suffices to consider the port numbering induced on **G** by γ^{-1}.

Finally:

Corollary 31 *A labelled graph* **G** *is minimal if and only if for any port numbering* δ, $(\mathbf{G}, \delta)$ *is minimal.*

From Lemma 27 we obtain:

Lemma 32 *For any minimal labelled graph it exists an election algorithm in Angluin's model.*

Conversely, the original impossibility result of Angluin [3] states that it is impossible to solve leader election in a non-minimal graph. Even if Angluin's proof does not use exactly the same coverings (she only considers simple graphs), Proposition 11 remains true in Angluin's model. Indeed, with the proof schemata of Proposition 11 we prove that if a labelled graph **G** with the port numbering δ is a covering of a labelled graph **H** with the port numbering δ', then in Angluin's model for any algorithm $\mathcal{A}$ there exists a run of $\mathcal{A}$ on $(\mathbf{G}, \delta)$ obtained from a run on $(\mathbf{H}, \delta')$. Thus for any graph **G** that is not minimal there exists a port numbering δ of **G** such that there is no election algorithm and no enumeration algorithm for $(\mathbf{G}, \delta)$ in Angluin's model. This fact and Lemma 32 prove:

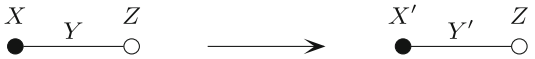
Theorem 33 *For any simple labelled graph* **G** *there exists an enumeration or an election algorithm for* **G** *in Angluin's model if and only if* **G** *is minimal.*

Remark 34 As for local computations on labelled edges, the graph **G** is given, thus each vertex knows the size of the graph.

7 Local computations on labelled edges and local computations on open labelled edges are equivalent

In this part, we prove that an algorithm described by local computations on labelled edges, i.e., of the form:

where $X' = f_1(X, Y, Z)$, $Y' = f_2(X, Y, Z) = f_2(Z, Y, X)$, $Z' = f_3(Z, Y, X)$, f_1, f_2 and f_3 are transition functions on triple of states can be simulated by an algorithm described by local computations on open labelled edges, i.e., of the form:



where $X' = f_1(X, Y, Z)$ and $Y' = f_2(X, Y, Z)$.

The converse is obvious. The proof of this property is not so simple. For example, let us consider the problem consisting of naming the edges such that each vertex does not have two incident edges with the same name. It is easy to find a simple algorithm using local computations on labelled edges that solves this problem, by using a rule like $\mathcal{M}_4$ of the enumeration algorithm for example, whereas it does not seem so easy to find an algorithm using local computations on open labelled edges that solves the same problem.

First we define formally the notion of simulation.

Definition 35 Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be two algorithms working on label sets L_1 and L_2. The algorithm $\mathcal{R}_2$ simulates the algorithm $\mathcal{R}_1$ if there exist two mappings ι from L_1 to L_2 and π from L_2 to L_1 such that for each labelled graph $\mathbf{G} = (G, \lambda)$:

- if each run of $\mathcal{R}_1$ on $\mathbf{G}$ terminates then each run of $\mathcal{R}_2$ on $(G, \iota \circ \lambda)$ terminates, and
- if for each irreducible labelled graph (G, λ_2') obtained from $(G, \iota \circ \lambda)$ with respect to $\mathcal{R}_2$ there exists an irreducible graph (G, λ_1') obtained from (G, λ) with respect to $\mathcal{R}_1$ such that $\pi \circ \lambda_2' = \lambda_1'$.

7.1 The main ideas for the simulation

We give a way to simulate a distributed algorithm $\mathcal{R}$ described by local computations on labelled edges by a distributed algorithm $S(\mathcal{R})$ described by local computations on open labelled edges. A distributed algorithm $\mathcal{R}$ using local computations on labelled edges, can be described by a relabelling system $\{r_i = (\lambda_i, \lambda_i')\}_{i \geq 0}$ of relabelling rules where each λ_i and λ_i' are two labellings of a graph $A = (\{v_1, v_2\}, f)$ such that $\mathrm{Ends}(f) = \{v_1, v_2\}$.

For each relabelling rule r_i in $\mathcal{R}$, we define below five relabelling rules and we will show later that the algorithm $S(\mathcal{R})$ described by these new relabelling rules is a simulation of $\mathcal{R}$ that uses local computations on open labelled edges.

A vertex can have the status **free**, **ask** or **acc**; if it is **ask**, it means it wants to simulate a relabelling step of $\mathcal{R}$ with one of its neighbours; if it is **acc**, it means it has accepted to simulate a relabelling step of $\mathcal{R}$ and it is waiting for an acknowledgement from its neighbour involved in this step simulation. An edge can be **free** if it is not involved in a relabelling step simulation, otherwise, it can be in an **ask** state which means that one of its endvertices wants to simulate a relabelling step of $\mathcal{R}$ with the other, or in a **acc** state, which means that an endvertex has accepted to simulate a relabelling step of $\mathcal{R}$ with the other one, or in an **ack** state, which means that the endvertex that wanted to simulate the step has been informed that the other one has accepted.

A vertex v can be in an **ask** state because it wants to simulate a relabelling step with one of its neighbours v', but this vertex v' can have simulated a relabelling step with another neighbour and therefore its label has changed: in this case, a rule enables an **ask** node to get its former label back.

Notation In the sequel, the rule r defined by:

will be also denoted by $r = (X, Y, Z, X', Y', Z')$.

7.2 Labels for the simulation

We assume that the label $\perp$ is not used by $\mathcal{R}$.

7.2.1 Vertices

Each vertex v has the label $(\lambda(v), \textbf{status}(v))$ where:

- $\lambda(v)$ is the label of v in $\mathcal{R}$;
- $\textbf{status}(v) \in \{\textbf{free}, \textbf{ask}, \textbf{acc}\}$ is the status of v, it depends on the state of the simulation.

The initial label of each vertex v is $(\lambda(v), \textbf{free})$.

7.2.2 Edges

Each edge e has the label $(\lambda(e), \textbf{status}(e), r(e))$ where:

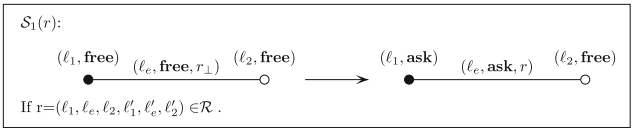
- $\lambda(e)$ is the label of e in $\mathcal{R}$,
- $\textbf{status}(e) \in \{\textbf{free}, \textbf{ask}, \textbf{acc}, \textbf{ack}\}$ is the status of e and corresponds to the different states of the simulation,
- $r(e) = (\ell_1, \ell_e, \ell_2, \ell_1', \ell_e', \ell_2')$ is the relabelling rule that $S(\mathcal{R})$ tries to simulate over e; if the status of e is **free**, then $r(e) = r_\perp = (\perp, \perp, \perp, \perp, \perp, \perp)$.

Initially, each edge e is labelled $(\lambda(e), \textbf{free}, r_\perp)$.

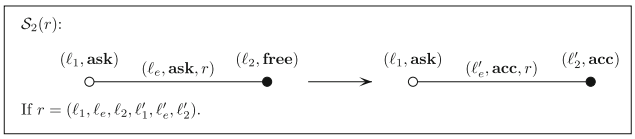
7.3 Relabelling rules for the simulation

In this part we give relabelling rules which enable the simulation of local computations on labelled edges by local computations on open labelled edges. We recall that in relabelling rules below, vertices filled with black (resp. unfilled vertices) is only a notation to underline that they are active (resp. passive) and it has no influence on the relabelling.

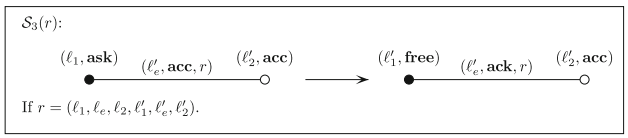
We follow the informal description given below. If the status of the endvertices of an edge is **free** and if a rule r of $\mathcal{R}$ may be applied on this edge then the status of an endvertex of this edge and the status the edge itself become **ask**. We memorize in the state of the edge the relabelling rule $S(\mathcal{R})$ tries to simulate. The first rule encodes this situation.

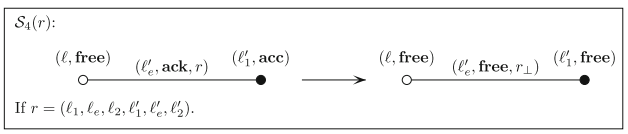


If a vertex v has a neighbour v' which has applied the rule $\mathcal{S}_1(r)$, if the status of v is **free** and if the label $\lambda(v)$ has not been modified then v may apply the following rule: the status of v and of e become **acc** and the labels $\lambda(v)$ and $\lambda(e)$ are modified.

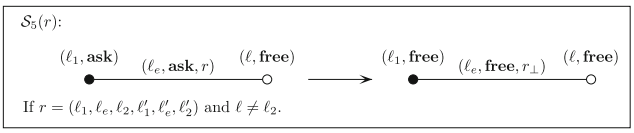


The two next rules correspond to the end of the application of the simulated rule.





The last rule enables a vertex v to become **free** if the vertex v' which has enabled the status **ask** for the vertex v has been relabelled with another vertex and v' cannot be changed any more.



7.4 Proof of the simulation

Let L_1 be a label set. Let $\mathcal{R}$ be an algorithm described with local computations on labelled edges (its label set is L_1). Let $\mathbf{G} = (G, \lambda)$ be a labelled graph on L_1. Let S be the labelling transformation defined by $S(\mathbf{G}) = (G, \lambda')$ where:

- if v is a vertex of G labelled $\lambda(v) : \lambda'(v) = (\lambda(v), \mathbf{free})$;
- if e is an edge of G labelled $\lambda(e) : \lambda'(e) = (\lambda(e), \mathbf{free}, r_\perp)$.

Let C be the inverse transformation of S, i.e., C is such that $C(S(\mathbf{G})) = \mathbf{G}$. Let $S(\mathcal{R})$ be local computations on open labelled edges obtained from $\mathcal{R}$ by the transformations of the previous subsection.

Lemma 36 *Let $\mathcal{R}$ be a labelled edge locally generated relabelling relation. Let $\mathbf{G}$ be a labelled graph. Let $(\mathbf{G}_i)_{0\leq i\leq n}$ be a relabelling sequence obtained from $\mathbf{G}_0 = \mathbf{G}$ with respect to $\mathcal{R}$. There exists a relabelling sequence $(\mathbf{G'}_i)_{0\leq i\leq n'}$ obtained from $\mathbf{G'}_0 = S(\mathbf{G}_0)$ with respect to $S(\mathcal{R})$ such that $C(\mathbf{G}'_{n'}) = \mathbf{G}_n$. Furthermore $\mathbf{G}_n$ is irreducible if and only if $\mathbf{G}'_n$ is irreducible.*

Proof It suffices to show the claim for $n = 1$, i.e., if $\mathbf{G}_0 \underset{\mathcal{R}}{\Longrightarrow} \mathbf{G}_1$ then there exists $\mathbf{G}'_1$ such that $S(\mathbf{G}_0) \underset{S(\mathcal{R})}{\overset{*}{\Longrightarrow}} \mathbf{G}'_1$ and $C(\mathbf{G}'_1) = \mathbf{G}_1$.

If $\mathbf{G}_0 \underset{\mathcal{R}}{\Longrightarrow} \mathbf{G}_1$ then there exists a rule $r \in R$ such that $\mathbf{G}_0 \underset{r}{\Longrightarrow} \mathbf{G}_1$; let e be the edge over which r is applied.

Clearly, the sequence $S_1(r)$, $S_2(r)$, $S_3(r)$ and $S_4(r)$ can be applied on e in $S(\mathbf{G_0})$:

$$\mathbf{G}'_0 \underset{S_1(r)}{\Longrightarrow} \mathbf{I}_1 \underset{S_2(r)}{\Longrightarrow} \mathbf{I}_2 \underset{S_3(r)}{\Longrightarrow} \mathbf{I}_3 \underset{S_4(r)}{\Longrightarrow} \mathbf{G'}_1.$$

Thus we obtain the labelled graph $\mathbf{G}'_1$ verifying $C(\mathbf{G}'_1) = \mathbf{G}_1$. The last part of the lemma is obvious.

Let $G = (V, E)$ be a graph; let E' be a subset of E; the subgraph induced by E' is the graph $G' = (V', E')$ where $V' = \{v \in V | v \in Ends(e'), e' \in E'\}$.

Proposition 37 *Let $\mathcal{R}$ be a labelled edge locally generated relabelling relation. Let $\mathbf{G}$ be a labelled graph. Let $(\mathbf{G'}_i)_{0\leq i\leq n}$ be a relabelling sequence obtained from $\mathbf{G'}_0 = S(\mathbf{G})$ with respect to $S(\mathcal{R})$. For any i $(0 \leq i \leq n)$ there exists a labelled graph $\mathbf{K}$ that can be obtained from $\mathbf{G}'_i$ with respect to $S(\mathcal{R})$ such that $C(\mathbf{K})$ can be obtained from $\mathbf{G}$ with respect to $\mathcal{R}$, i.e., $\mathbf{G}'_\mathbf{i} \underset{S(\mathcal{R})}{\overset{*}{\Longrightarrow}} \mathbf{K}$ and $\mathbf{G} \underset{\mathcal{R}}{\overset{*}{\Longrightarrow}} C(\mathbf{K})$.*

Proof By induction on i. It is obvious for $i = 0$.

Let $\mathbf{F}_i$ be the subgraph of $\mathbf{G'}_i$ induced by edges having a status different from **free**. We prove that by applying rules of $S(\mathcal{R})$ on edges of $\mathbf{F}_i$ one can obtain a labelled graph $\mathbf{K}$ verifying properties of the proposition.

First, we prove that the graphs $\mathbf{G'}_i$ and $\mathbf{F}_i$ verify the following decomposition and properties.

1. A vertex of $\mathbf{G}'_i$ which does not belong to $\mathbf{F}_i$ has the status **free**.
2. The graph $\mathbf{F}_i$ is an union of disjoint trees: $\mathbf{F}_i = \cup_{1\leq j\leq k}\mathbf{T}_j$; each tree $\mathbf{T}_j$ contains either exactly one vertex with the status **free** or exactly one edge with the status **acc**.
   (a) If $\mathbf{T}_j$ contains a vertex, denoted by v, with the status **free** then the only relabelling rule that can be applied on $\mathbf{G'}_i$ and which intersects $\mathbf{T}_j$ must be applied on an edge incident to the vertex v.
      i. If we apply a relabelling rule on an edge labelled **free** incident to v, this edge is outside of $\mathbf{T}_j$ and the rule is of the form $\mathcal{S}_1(r)$ for

some rule r. Let $\mathbf{G}''$ be the graph we obtain from $\mathbf{G}'_i$.

ii. If we apply a rule inside $\mathbf{T}_j$. Let $v_1, v_2, \ldots, v_m$ be the neighbours of v in $\mathbf{T}_j$. let $\mathbf{T}_j^{(\ell)}$ be the tree defined by considering the connected component of $\mathbf{T}_j$ we obtain by the deletion of the edge $\{v, v_\ell\}$ and which contains v_ℓ, $1 \leq \ell \leq m$. Each edge $\{v, v_\ell\}$ has either the status **ask** or the status **ack**;

A. if the edge $\{v, v_\ell\}$ has the status **ask** then v_ℓ has the status **ask**; in this case the status of the edge will change either by the application of a rule of the form $\mathcal{S}_2(r)$ or by the application of a rule of the form $\mathcal{S}_5(r)$. In both cases, r is encoded in the label of the edge.

– If we apply a rule of the form $\mathcal{S}_2(r)$ then let $\mathbf{G}''$ be the labelled graph we obtain from $\mathbf{G}'_i$.

– If we apply a rule of the form $\mathcal{S}_5(r)$ then let $\mathbf{G}''$ be the graph we obtain from $\mathbf{G}'_i$.

B. If $\{v, v_\ell\}$ has the status **ack** then v_ℓ has the status **acc**, in this case the only rule that may be applied on this edge has the form $\mathcal{S}_4(r)$, where r is the rule encoded in the label of the edge. Let $\mathbf{G}''$ be the labelled graph we obtain by the application of $\mathcal{S}_4(r)$ on $\{v, v_l\}$.

(b) If $\mathbf{T}_j$ contains one edge, denoted by e, with the status **acc** then an endvertex of e, denoted by v_1, has the status **ask** and the other one, denoted by v_2, has the status **acc**. The only relabelling rule that can be applied on $\mathbf{G}'_i$ and which intersects $\mathbf{T}_j$ must be applied on the edge e; the application of this rule, having the form $\mathcal{S}_3(r)$ (for some r) over the edge e, creates a tree having exactly one vertex with the status **free**. Let $\mathbf{G}''$ be the labelled graph we obtain by applying the rule $\mathcal{S}_3(r)$ on e.

If $\mathbf{G}_{i+1} = \mathbf{G}''$ then $\mathbf{G}_{i+1}$ verifies the decomposition and the properties given above.

If $\mathbf{G}_{i+1}$ is obtained by applying a rule of the form $\mathcal{S}_1(r)$ outside of $\mathbf{F}_i$ then $\mathbf{G}_{i+1}$ also verifies the same decomposition and properties. Finally we have proved:

Property 1 If the decomposition and the properties given above hold at step i then it holds too at step $i + 1$.

Furthermore:

Property 2 Let v be a vertex with the status **free** which does not belong to $\mathbf{F}_{i+1}$. By definition, the edges which are incident to v have the status **free**. If v does not belong to $\mathbf{F}_i$ then no rule incident to v has been applied at step i. If v belongs to F_i then a rule has been applied to an edge incident to v, this edge obtains the status **free** thus its endvertices also obtain the status **free**.

To achieve the proof of the proposition, by a simple induction we verify the following assumptions (we consider once more time the decomposition and the properties described above):

1. if rules of $S(\mathcal{R})$ are applied to edges of $\mathbf{T}_j$ $(1 \leq j \leq k)$ then we obtain a graph K such that $\mathbf{G} \underset{\mathcal{R}}{\overset{*}{\Longrightarrow}} C(\mathbf{K})$;
2. if a tree $\mathbf{T}_j$ contains one vertex with the status **free** , denoted by v, then let $\mathbf{T}'_j$ be the tree we obtain by replacing the status of some edges incident to v in $\mathbf{T}_j$ by **free** and the status of the corresponding endvertex by **free**; for all combinations, if rules of $S(\mathcal{R})$ are applied to edges of $\mathbf{T}'_j$ $(1 \leq j \leq k)$ then we obtain a graph K such that $\mathbf{G} \underset{\mathcal{R}}{\overset{*}{\Longrightarrow}} C(\mathbf{K})$.

From the decompositions given in the proof of the previous proposition, we deduce the 3 following corollaries:

Corollary 38 *Let $\mathcal{R}$ be a labelled edge locally generated relabelling relation. Let $\mathbf{G} = (G, \lambda)$ be a labelled graph. Let $(\mathbf{G}'_i)_{0 \leq i \leq n}$ be a relabelling sequence obtained from $\mathbf{G}'_0 = S(\mathbf{G})$ with respect to $S(\mathcal{R})$. If a rule having the form $\mathcal{S}_1(r)$, for some r, is applied on the edge $\{v_1, v_2\}$ at step k of the relabelling sequence where the label of v_1 becomes* **ask** *and if the status of $\{v_1, v_2\}$ becomes* **free** *for the first time at the step k' $(k' > k)$ then at least one of the first component of the labels of v_1 or of v_2 or of $\{v_1, v_2\}$ has been changed since the step k. This change comes from the application of a sequence of relabelling rules of the form $\mathcal{S}_2(r')$, $\mathcal{S}_3(r')$ and $\mathcal{S}_4(r')$ for some rule r' applied either on the edge $\{v_1, v_2\}$ (in this case $r = r'$) or on an edge incident to the vertex v_2.*

Corollary 39 *Let $\mathcal{R}$ be a labelled edge locally generated relabelling relation. Let $\mathbf{G} = (G, \lambda)$ be a labelled graph. Let $(\mathbf{G}'_i)_{0 \leq i \leq n}$ be a relabelling sequence obtained from $\mathbf{G}'_0 = S(\mathbf{G})$ with respect to $S(\mathcal{R})$. The labelled graph $\mathbf{G}'_n$ is irreducible modulo $S(\mathcal{R})$ if and only if each vertex and each edge of $\mathbf{G}'_n$ has the status* **free** *and $C(\mathbf{G}'_n)$ is irreducible modulo $\mathcal{R}$.*

Corollary 40 *Let $\mathcal{R}$ be a labelled edge locally generated relabelling relation. If $\mathcal{R}$ has the termination property then $S(\mathcal{R})$ has the termination property.*

Proof By contradiction. If there exists a non bounded relabelling sequence with respect to $S(\mathcal{R})$ then necessary rules of the form $\mathcal{S}_2(r)$ are applied a non bounded number of times and from Corollary 38 and by Corollary 39 we deduce a non bounded relabelling sequence with $\mathcal{R}$.

Finally from Lemma 36 and these corollaries:

Theorem 41 *Let $\mathcal{R}$ be a labelled edge locally generated relabelling relation. The relabelling system $S(\mathcal{R})$ simulates $\mathcal{R}$.*

Corollary 42 *Let* **G** *be a labelled graph. There exists an election algorithm for* **G** *based on local computation on labelled edges if and only if there exists an election algorithm for* **G** *based on local computations on open labelled edges.*

8 Application to asynchronous systems with synchronous message passing

We refer to the definition given by Tel [27, p. 47]: a message passing is said to be synchronous if a send event and the corresponding receive event are coordinated to form a single transition of the system, i.e., a transition is an atomic event corresponding to both the receive and send events. This hypothesis implies that, as for Angluin's model, the initial symmetry between two adjacent vertices can be non-deterministically broken.

Finally, if we consider an asynchronous system with a port numbering and synchronous message passing, events may be encode by rules having the form given in Figs. 12, 13.

We prove in the sequel that this model is equivalent to the model of local computations on labelled edges with a port numbering and consequently to the model of Angluin. By this way we characterize graphs that admit an election algorithm in this model (let **G** be a labelled network, as for Angluin's model we ask that there exists an election algorithm for every port numbering δ of **G**.).

By definition, Angluin's model is more powerful than the synchronous message passing model.

In the case of synchronous message passing with a port numbering we cannot memorize the label of the edge $e = \{u, v\}$ on the two endvertices u and v simultaneously: only one endvertex may be relabelled (with notation of the rule R_{Synch} it corresponds to $Z' = f(Z, X, j)$ on the vertex v). We memorize the new state of the edge on the vertex v associated with the port j. The problem consists, when the next

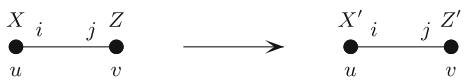


Fig. 12 The rule R_{Synch} where $X' = f_1(X, i)$, $Z' = f_2(Z, X, j)$, i and j are the corresponding port numbers and f_1, f_2 are transition functions

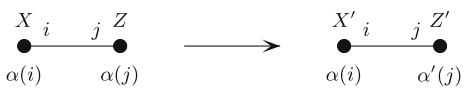


Fig. 13 We assume that $\alpha(i)$ and $\alpha(j)$ are states associated to the edge through ports i and j; the state up to date of the edge e is $\alpha'(j)$

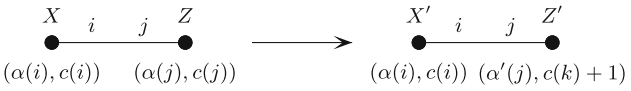


Fig. 14 The new state of the edge is memorized on the vertex v and its counter is $c(k) + 1$

synchronization will happen on the edge e, in the determination of the vertex on which is memorized the last state associated to e examining the states of u and v. Consider the following notation:

A natural solution with this problem is obtained by associating to the state of the edge memorized on the endvertices counters $c(i)$ and $c(j)$ (the initial value is 0) and each time there is a transformation on the edge e the state of e corresponds to the state on u or v associated to the counter having the maximal value. When a transformation is applied on the edge, it computes the new state of the edge, adds 1 to the counter having the maximal value and associates this new value of the counter to the new state of e on v. For the rule R_{Synch}, let k such that $c(k) = Max\{c(i), c(j)\}$, before the application of R_{Synch} the state of the edge is $\alpha(k)$. After the application of R_{Synch} the new state of e is $\alpha'(j)$ and its counter is $c(k) + 1$ (see Fig. 14).

In fact we can solve this problem with only 3 values $\{0, 1, 2\}$ by applying the following method: if for a endvertex the counter is 0 and it is 1 for the other endvertex then the maximal counter is 1; if it is 1 and 2 then the maximal counter is 2; if it is 0 and 2 then the maximal counter is 0. If the state up to date of the edge (the state of the edge associated to the counter having the maximal value) before the application of the transformation is on the receiver vertex then the new counter associated to the new state of the edge is equal to the old; if not, to obtain the value of the new counter associated with the new state of the edge, we increment modulo 3 the maximal counter before the transformation.

That proves that one can simulate any local computation on open labelled edges by synchronous message passing.

Finally, from Theorem 41:

Theorem 43 *There exists an enumeration or an election algorithm in an asynchronous network* **G** *with synchronous message passing if and only if* **G** *is minimal.*

9 Comparison of the power of the previous models

Angluin's model and synchronous message passing model consider port numbering and thus they use as initial knowledge the vertex-degrees (i.e., for each vertex its initial label contains its degree). Results of previous sections imply:

Theorem 44 *The following models are equivalent:*

1. *local computations on labelled edges with the knowledge of vertex-degrees,*

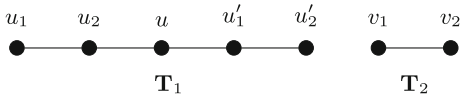


Fig. 15 The graphs $\mathbf{T}_1$ and $\mathbf{T}_2$ are minimal, nevertheless it does not exist local computations on labelled edges without the knowledge of the vertex-degrees which solves the election problem on the family $\{\mathbf{T}_1, \mathbf{T}_2\}$

2. *local computations on open labelled edges with the knowledge of vertex-degrees,*
3. *Angluin's model,*
4. *synchronous message passing model.*

One can wonder whether the knowledge of the vertex-degrees makes local computations on labelled edges more powerful. We give in the sequel a positive answer to this question. For that, we consider the election problem for families of labelled graphs.

Proposition 45 *Let $\mathbf{T}_1$ and $\mathbf{T}_2$ given in Fig. 15. It does not exist an election algorithm $\mathcal{R}$ defined by local computations on labelled edges without the knowledge of vertex-degrees for the family $\{\mathbf{T}_1, \mathbf{T}_2\}$.*

Proof Let $\mathcal{R}$ be an election algorithm defined by local computations on labelled edges which solves the election problem for $\mathbf{T}_2$. We consider a run ρ of $\mathcal{R}$ on $\mathbf{T}_2$ and we assume that v_1 is the elected vertex. We consider the subgraph $\mathbf{T}$ (resp. $\mathbf{T}'$) of $\mathbf{T}_1$ induced by vertices u_1 and u_2 (resp. u'_1 and u'_2). As $\mathbf{T}$ and $\mathbf{T}'$ are isomorphic to $\mathbf{T}_2$, there exists a run of $\mathcal{R}$ on $\mathbf{T}$ and $\mathbf{T}'$ such that u_1 and u'_1 are elected. We get a contradiction.

However, from the election algorithm given in Fig. 5 we deduce:

Lemma 46 *There exists an election algorithm defined by local computations on labelled edges for the family of trees when each vertex initially knows its degree.*

Thus:

Proposition 47 *Local computations on labelled edges with the knowledge of vertex-degrees are strictly more powerful than local computations on labelled edges without the knowledge of vertex-degrees.*

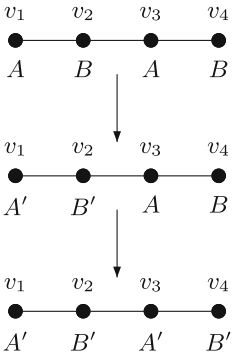
10 More comparisons

10.1 Is it important to have labels on edges

In our models, we have considered labelled graphs such that the edges can have labels and this property has been used to describe the different algorithms we present. We wonder if the results remain true when we consider models where the edges cannot be labelled. We present a minimal graph in which we cannot find an election algorithm using local computations on edges when the edges are not labelled.

Fig. 16 A relabelling rule

Fig. 17 Application of a relabelling rule



Consider a path $\mathbf{G}$ on four vertices $\{v_1, v_2, v_3, v_4\}$ (for each $1 \leq i \leq 3$, v_i is a neighbour of v_{i+1}) where all vertices have the same initial label. This graph is minimal and therefore we can solve the election problem with local computations on labelled edges. Consider a relabelling relation $\mathcal{R}$ having the termination property and associated to an algorithm involving local computations on labelled edges such that there is not any rule that labels the edges.

Consider Fig. 17. We prove by induction that there exists an execution of $\mathcal{R}$ such that the vertices v_1 and v_3 (resp. v_2 and v_4) have the same labels. Initially, the result is true since all vertices have the same initial label. If at a step $i+1$, a rule R is applied, this rule has necessarily the form given in Fig. 16, where A is the label of v_1 and B is the label of v_2. As described in Fig. 17, the rule R can be applied to the nodes v_1 and v_2 and then to the nodes v_3 and v_4: the property holds.

10.2 Local computations on edges without port numbering (labelling)

This kind of computations may be illustrated by population protocols inspired by sensor networks and defined by Angluin et al. in [1,2,4]. These models are based on pairwise interactions of mobile agents in populations. In such interactions, there is an initiator and a responder.

10.2.1 Local computations on open edges

This case considers only one-way communication between two agents in an interaction. The state of the initiator remains unchanged, only the state of the responder changes. It corresponds to Fig. 18.

In [9], a complete characterization of graphs for which enumeration and election are possible is presented. We can

Fig. 18 We assume that $X' = f(X, Z)$ and f is a transition function

Fig. 19 We assume that $X' = f_1(X, Z)$, $Z' = f_2(Z, X)$ and f_1, f_2 are transition functions

notice that the class of graphs for which enumeration is solvable is different from the class of graphs for which election is solvable.

10.2.2 Local computations on edges

In this case we assume that two-way communication is possible. The two agents play distinct roles thus the model is asymmetric (see Fig. 19).

Graphs for which enumeration and election are possible are characterized in [6].

Remark 48 In [1,2] a finite set of possible states for an agent is given it defined the set of values of the memory of each agent.

11 Conclusion

In this work, our initial motivation was to study local computations on labelled edges as a simple and natural example of local computations on labelled graphs which encode synchronism in distributed computing. The study of the impact of the synchronism on the computational power is an old and natural question; it has been done also for the classical consensus problem (see for example [11]).

The characterization of graphs that admit a naming (an election) algorithm in this model is a characterization of the same kind as other results existing for different models [5,6,20]. For all these models, one can find a particular type of locally constrained homomorphisms such that the graphs where we can solve the naming problem are the graphs that are minimal according to this type of locally constrained homomorphisms.

We have also proved the non trivial equivalence between local computations on labelled edges and local computations on open labelled edges.

As nice corollaries of our study we have obtained (for the first time to our knowledge) the characterization of graphs that admit and election algorithm for two seminal models : Angluin's model and the atomic receive/send model, which are considered as realistic model to understand some paradigms of distributed computations.

Our study enables also to have a better understanding of some initial knowledges as the vertex-degrees or the importance of labels on edges.

In Mazurkiewicz's model, the algorithm presented by Mazurkiewicz in [20] is used as a basic building block to solve other classical problems in distributed computing. In [16], Godard et al. characterize graph classes that can be recognized in a distributed way. In [23], Métivier and Tel characterize graph classes where any distributed algorithm can be transformed into a distributed algorithm where processes can detect that the global computation is finished and in [15], Godard and Métivier characterize graph classes that admit an election algorithm.

It is a natural question to wonder if these results can be extended to the local computations on labelled edges, once we have a Mazurkiewicz-like algorithm in this model.

References

1. Angluin, D., Aspnes, J., Diamadi, Z., Fisher, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distr. Comput. **18**(4), 235–253 (2006)
2. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: On the power of anonymous one-way communication. In: Proceedings of 9th Conference on Principles of Distributed Computing, pp. 307–318 (2005)
3. Angluin, D.: Local and global properties in networks of processors. In: Proceedings of the 12th Symposium on Theory of Computing, pp. 82–93 (1980)
4. Aspnes, J., Ruppert, E.: An introduction to population protocols. In: Middleware for Network Eccentric and Mobile Applications, pp. 97–119. Springer, Berlin (2009)
5. Boldi, P., Codenotti, B., Gemmell, P., Shammah, S., Simon, J., Vigna, S.: Symmetry breaking in anonymous networks: Characterizations. In: Proceedings of 4th Israeli Symposium on Theory of Computing and Systems, pp. 16–26. IEEE Press (1996)
6. Chalopin, J.: Local computations on closed unlabelled edges: The election problem and the naming problem. In: Proceedings of the 31st Conference on Current Trends in Theory and Practice of Informatics (SOFSEM 2005), vol. 3381 of Lecture Notes in Computer Science, pp. 81–90. Springer, Berlin (2005)
7. Chalopin, J., Métivier, Y.: Election and local computations on edges. In: Proceedings of Foundations of Software Science and Computation Structures, 7th International Conference (FOSSACS 2004), vol. 2987 of *Lecture Notes in Computer Science*, pp. 90–104. Springer, Berlin (2004)
8. Chalopin, J., Métivier, Y.: An efficient message passing election algorithm based on mazurkiewicz's algorithm. Fundam. Inf. **80**(1-3), 221–246 (2007)
9. Chalopin, J., Métivier, Y., Zielonka, W.: Local computations in graphs: The case of cellular edge local computations. Fundam. Inf. **74**(1), 85–114 (2006)
10. Chalopin, J., Paulusma, D.: Graph labelings derived from models in distributed computing. In: WG, pp. 301–312 (2006)
11. Dolev, D., Dwork, C., Stockmeyer, L.J.: On the minimal synchronism needed for distributed consensus. J. ACM **34**(1), 77–97 (1987)
12. Fiala, J., Paulusma, D.: A complete complexity classification of the role assignement problem. Theor. Comput. Sci. **349**, 67–81 (2005)
13. Fiala, J., Paulusma, D., Telle, J.A.: Locally constrained graph homomorphisms and equitable partitions. Eur. J. Comb. **29**(4), 850–880 (2008)

14. Ghosh, S.: Distributed systems—an algorithmic approach. Chapman and Hall/CRC, London (2006)
15. Godard, E., Métivier, Y.: A characterization of families of graphs in which election is possible (*ext. abstract*). In Nielsen, M., Engberg, U. (eds), Proceedings of Foundations of Software Science and Computation Structures, FOSSACS'02, number 2303 in LNCS, pp. 159–171. Springer, Berlin (2002)
16. Godard, E., Métivier, Y., Muscholl, A.: Characterization of classes of graphs recognizable by local computations. Theory Comput. Syst. **37**(2), 249–293 (2004)
17. Godard, E.: A self-stabilizing enumeration algorithm. Inf. Process. Lett. **82**(6), 299–305 (2002)
18. Hoare, C.A.R.: Communicating sequential processes. Commun. ACM **21**(8), 666–677 (1978)
19. Mazurkiewicz, A.: Trace theory. In: Brauer, W. et al., (eds.) Petri nets, applications and relationship to other models of concurrency. vol. **255** of Lecture notes in computer science, pp. 279–324. Springer, Berlin (1987)
20. Mazurkiewicz, A.: Distributed enumeration. Inf. Process. Lett. **61**, 233–239 (1997)
21. Mazurkiewicz, A.: Bilateral ranking negotiations. Fundam. Inf. **60**(1–4), 1–16 (2004)
22. Milne, G., Milner, R.: Concurrent processes and their syntax. J. ACM **26**(2), 302–321 (1979)
23. Métivier, Y., Tel, G.: Termination detection and universal graph reconstruction. In *SIROCCO 00—7th International Colloquium on Structural Information & Communication Complexity*, pp. 237–251 (2000)
24. Reidemeister, K.: Einführung in die Kombinatorische Topologie. Vieweg, Brunswick (1932)
25. Rosen, K.H. (ed.): Handbook of discrete and combinatorial mathematics. CRC Press, Boca Raton (2000)
26. Tanenbaum, A., Steen, M.van : Distributed systems—principles and paradigms. Prentice Hall, London (2002)
27. Tel, G.: Introduction to distributed algorithms. Cambridge University Press, Cambridge (2000)
28. Yamashita, M., Kameda, T.: Computing on anonymous networks: Part i—characterizing the solvable cases. IEEE Transactions on Parallel and Distributed Systems **7**(1), 69–89 (1996)
29. Yamashita, M., Kameda, T.: Leader election problem on networks in which processor identity numbers are not distinct. IEEE Trans. Parallel Distrib. Syst. **10**(9), 878–887 (1999)