Graph Relabelling Systems : a Tool for Encoding, Proving and Studying Distributed Algorithms

Yves Métivier

LaBRI, UMR CNRS 5800
Université Bordeaux I - E.N.S.E.R.B. - IUF
345, cours de la Libération
33405 Talence, France
Yves.Metivier@labri.u-bordeaux.fr

Abstract. Graph relabelling systems have been introduced as a suitable model for expressing and studying distributed algorithms on a network of communicating processors. We recall the basic ideas underlying that model and we survey the main questions that have been considered and the main results that have been obtained in that framework.

Keywords : distributed algorithm, covering, election, local computations in graphs, recognition, spanning tree, termination detection.

1 Introduction

Graph relabelling systems and, more generally, local computations in graphs are powerful models which provide general tools for encoding distributed algorithms, for proving their correctness and for understanding their power. We consider a network of processors with arbitrary topology. It is represented as a connected, undirected graph where vertices denote processors, and edges denote direct communication links. An algorithm is encoded by means of local relabelings. Labels attached to vertices and edges are modified *locally*, that is on a subgraph of fixed radius k of the given graph, according to certain rules depending on the subgraph only ($k-$*local computations*). The relabelling is performed until no more transformation is possible. The corresponding configuration is said to be in normal form.

The present contribution reflects classical topics including basic properties of local computations [16]. Among paradigms associated with local computations, we present the computation of a spanning tree, the election problem, the recognition problem and the local detection of the termination problem.

For these four problems, we consider graphs with an initial labelling which may encode identity or some knowledge on the graph as, for instance, the number of vertices and/or edges. For the recognition problem, the presence or the absence of certain final labels determines whether G is accepted or not. The aim of an election algorithm is to choose exactly one element among the set of vertices. We consider local computation systems such that for each irreducible graph there is a given vertex label which appears exactly once in the graph. A distributed algorithm terminates whenever it reaches a terminal configuration, that is a configuration in which no step of the algorithm can be applied. We are interested in the question whether the global termination of a system of local computations can be detected also locally. This means that for every terminal configuration there is a vertex in the graph such that its neighbourhood of a given radius r determines that a normal form has been reached. In this case, we say that global termination is $r-$*locally detected*.

We use coverings and quasi-coverings as fundamental tools which enable to understand the borderline between positive and negative results about distributed computations.

Many problems have no solution in distributed computing [18]. The introduction of randomization makes possible tasks that admit no deterministic solutions as, for instance, the synchronization problem or the election problem in an anonymous network. In the last section we present randomized algorithms which have been proposed and studied in [26, 27]. These randomized algorithms may be considered as basic steps for the implementation of local computations in an anonymous asynchronous system where processors communicate with asynchronous message passing. General considerations about randomized distributed algorithms may be found in [35] and some techniques used in the design and for the analysis of randomized algorithms are presented in [30, 20, 10].

Among models related to our model there are local computations systems, as defined by Rosensthiel *et al.* [34], Angluin [1] or Yamashita and Kameda [11, 12]. In [34] a synchronous model is considered, where all vertices are equipped with deterministic finite automata (the same for all vertices). A basic computation step consists then in computing the next state of each processor according to its own state and to the states of its

neighbours. In [1] an asynchronous model is considered: during a basic computation step, two adjacent vertices exchange their labels and then compute new labels. In [11, 12] an asynchronous model is also considered where during a basic computation step a processor either changes its state and sends a message or receives a message.

Introduction to distributed algorithms and main topics of the field are presented in [2, 19, 35, 31, 32].

2 Basic Definitions, Notation and Examples

2.1 A First Example

Let us first illustrate graph relabelling systems by considering a simple distributed algorithm which computes a spanning tree of a network. Assume that a unique given processor is in an "active" state (encoded by the label **A**), all other processors being in some "neutral" state (label **N**) and that all links are in some "passive" state (label **0**). The tree initially contains the unique active vertex. At any step of the computation, an active vertex may activate one of its neutral neighbours and mark the corresponding link which gets the new label **1**. This computation stops as soon as all the processors have been activated. The spanning tree is then obtained by considering all the links with label **1**. Fig. 1 describes a sample computation using this algorithm.

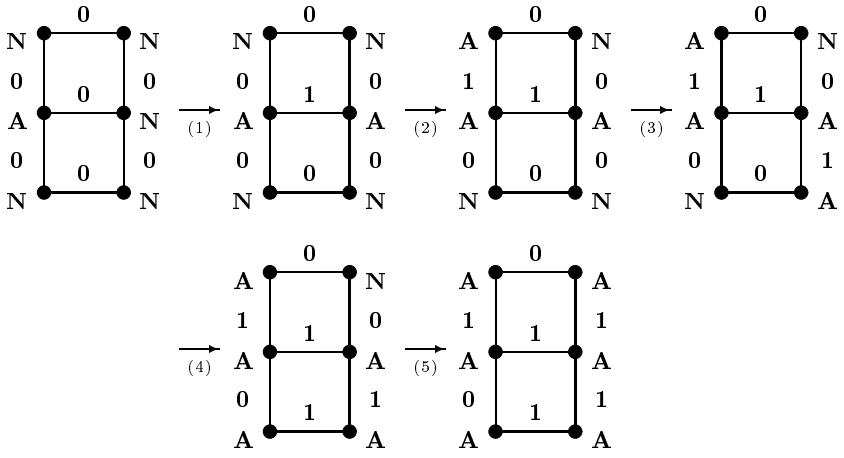


Fig. 1. Computation of a spanning tree.

An elementary step in this computation may be depicted as a *relabelling step* by means of the following relabelling rule R which describes the corresponding label modifications:

An application of this relabelling rule on a given graph (or network) consists in (i) finding in the graph a subgraph isomorphic to the left-hand-side of the rule (this subgraph is called the *occurrence* of the rule) and (ii) modifying its labels according to the right-hand-side of the rule.

The *relabelling sequence* depicted in Fig. 1 illustrates a *sequential* computation since the relabelling steps are sequentially applied. A *distributed* view of this computation can be obtained by considering that relabelling steps concerning *disjoint* parts of the graph may be applied in any order, or even concurrently (this is namely the case for the steps (2) and (3), or (4) and (5) in Fig. 1).

2.2 Definitions and Examples

(For further details on material about discrete and combinatorial mathematics see [33]). Unless otherwise stated, all the graphs considered in this paper are finite, undirected, without multiple edges, loopless and *connected*. For every graph G we denote by $V(G)$ its set of vertices and by $E(G)$ its set of edges. If G and G' are two graphs, we say that G' is a *subgraph* of G if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. If X is a subset of $V(G)$, the subgraph of G *induced* by X has vertex set X and edge set the set of all edges whose both extremities belong to X. A *homomorphism* of a graph G to a graph H is a mapping φ from $V(G)$ to $V(H)$ such that $\varphi(x)\varphi(y)$ is an edge in H whenever xy is an edge in G. We say that φ is an *isomorphism* if φ is bijective and φ^{-1} is also a homomorphism. In the following, a set of graphs which is closed under isomorphism will be called a *class* of graphs.

Let $\mathcal{L}$ be a set whose elements are called *labels*. A *$\mathcal{L}$-labelled graph* is a pair (G, λ) where G is a graph and λ a mapping from $V(G) \cup E(G)$ to $\mathcal{L}$. If (G, λ) and (G', λ') are two labelled graphs, we say that (G', λ') is a (labelled) subgraph of (G, λ) if G' is a subgraph of G and λ' is the restriction of λ to $V(G') \cup E(G')$. We will denote by $\mathcal{G}_\mathcal{L}$ the set of all $\mathcal{L}$-labelled graphs. An isomorphism between two labelled graphs (G, λ) and (H, μ) is an isomorphism φ between G and H which preserves the labels, that is $\lambda(x) = \mu(\varphi(x))$ for every x in $V(G)$ and $\lambda(xy) = \mu(\varphi(x)\varphi(y))$ for every xy in $E(G)$. An *occurrence* of (G, λ) in (H, μ) is an isomorphism φ between G and a subgraph (H', μ') of (H, μ). We will then write $\varphi(G, \lambda) = (H', \mu')$.

A *(graph) relabelling rule* is a triple $R = (G_R, \lambda_R, \lambda'_R)$ such that (G_R, λ_R) and (G_R, λ'_R) are two labelled graphs. The labelled graph (G_R, λ_R) (resp. (G_R, λ'_R)) is called the *left-hand side* (resp. *right-hand side*) of R.

A *graph relabelling system* (GRS for short) is a triple $\mathcal{R} = (\mathcal{L}, \mathcal{I}, P)$ where $\mathcal{L}$ is a set of labels, $\mathcal{I}$ a subset of $\mathcal{L}$ called the set of *initial labels* and P a finite set of relabelling rules. A *$\mathcal{R}$-relabelling step* is a 5-tuple $(G, \lambda, R, \varphi, \lambda')$ such that R is a relabelling rule in P and φ is both an occurrence of (G_R, λ_R) in (G, λ) and an occurrence of (G_R, λ'_R) in (G, λ'). Intuitively speaking, the labelling λ' of G is obtained from λ by modifying all the labels of the elements of $\varphi(G_R, \lambda_R)$ according to the labelling λ'_R. Such a relabelling step will be denoted by $(G, \lambda) \longrightarrow_{R,\varphi} (G, \lambda')$. A *$\mathcal{R}$-relabelling sequence* is a tuple $(G, \lambda_0, R_0, \varphi_0, \lambda_1, R_1, \varphi_1, \lambda_2, \ldots, \lambda_{n-1}, R_{n-1}, \varphi_{n-1}, \lambda_n)$ such that for every i, $0 \leq i < n$, $(G, \lambda_i, R_i, \varphi_i, \lambda_{i+1})$ is a $\mathcal{R}$-relabelling step. The existence of such a relabelling sequence will be denoted by $(G, \lambda_0) \longrightarrow^*_\mathcal{R} (G, \lambda_n)$.

A labelled graph (G, λ) is said to be *$\mathcal{R}$-irreducible* if there exists no occurrence of (G_R, λ_R) in (G, λ) for every relabelling rule R in P. For every labelled graph (G, λ) in $\mathcal{G}_\mathcal{I}$ we denote by $Irred_\mathcal{R}(G, \lambda)$ the set of all $\mathcal{R}$-irreducible labelled graphs (G, λ') such that $(G, \lambda) \longrightarrow^*_\mathcal{R} (G, \lambda')$. Intuitively speaking, the set $Irred_\mathcal{R}(G, \lambda)$ contains all the final labellings that can be obtained from a $\mathcal{I}$-labelled graph (G, λ) by applying relabelling rules in P and may be viewed as the set of all the possible results of the computation encoded by the system $\mathcal{R}$.

Example 1. The algorithm presented in the introduction may be encoded by the graph relabelling system $\mathcal{R}_1 = (\mathcal{L}_1, \mathcal{I}_1, P_1)$ defined by $\mathcal{L}_1 = \{\mathbf{N}, \mathbf{A}, \mathbf{0}, \mathbf{1}\}$, $\mathcal{I}_1 = \{\mathbf{N}, \mathbf{A}, \mathbf{0}\}$, and $P_1 = \{R\}$ where R is the following relabelling rule:

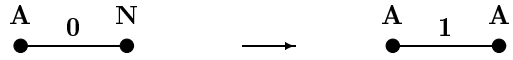
R: 

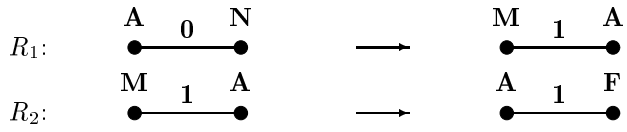
Fig. 1 describes a sample $\mathcal{R}_1$-relabelling sequence.

The notion of relabelling sequence defined above obviously corresponds to a notion of *sequential* computation. We can define a more distributed way of computing by saying that two relabelling steps concerning "disjoint" occurrences may be applied in any order, or even concurrently. It is easy to check that if $(G, \lambda_i, R_i, \varphi_i, \lambda_{i+1})$ and $(G, \lambda_{i+1}, R_{i+1}, \varphi_{i+1}, \lambda_{i+2})$ are two labelling steps such that $\varphi_i(G)$ and $\varphi_{i+1}(G)$ do not intersect then $(G, \lambda_i, R_{i+1}, \varphi_{i+1}, \lambda')$ and $(G, \lambda', R_i, \varphi_i, \lambda_{i+2})$ are two relabelling steps leading to the same resulting labelled graph (G, λ_{i+2}). More generally, any two relabelling sequences such that the latter one may be obtained from the former one by a succession of such "commutations" lead to the same resulting graph. Hence, our notion of relabelling sequence may be regarded as a *serialization* [22] of some distributed computation. This model is clearly asynchronous: several relabelling steps *may* be done at the same time but we do not demand all of them to be done. In the sequel we will essentially deal with sequential relabelling sequences but the reader should keep in mind that such sequences may be done in a distributed way.

In order to reach a satisfactory expressive power, we introduce some *local control mechanisms*. These mechanisms allow us to restrict in some sense the applicability of relabelling rules.

A graph relabelling system *with priorities* (PGRS for short) is a 4-tuple $\mathcal{R} = (\mathcal{L}, \mathcal{I}, P, >)$ such that $(\mathcal{L}, \mathcal{I}, P)$ is a graph relabelling system and $>$ is a partial order defined on the set P called the *priority relation*. A *$\mathcal{R}$-relabelling step* is then defined as a 5-tuple $(G, \lambda, R, \varphi, \lambda')$ such that R is a relabelling rule in P, φ is both an occurrence of (G_R, λ_R) in (G, λ) and an occurrence of (G_R, λ'_R) in (G, λ') and there exists no occurrence φ' of a relabelling rule R' in P with $R' > R$ such that $\varphi(G_R)$ and $\varphi(G_{R'})$ intersect in G (that is $V(\varphi(G_R)) \cap V(\varphi(G_{R'})) = \emptyset$). The notion of relabelling sequence is defined as previously.

Example 2. Let $\mathcal{R}_2 = (\mathcal{L}_2, \mathcal{I}_2, P_2, >_2)$ be the PGRS defined by $\mathcal{L}_2 = \{\mathbf{N}, \mathbf{A}, \mathbf{M}, \mathbf{F}, \mathbf{0}, \mathbf{1}\}$, $\mathcal{I}_2 = \{\mathbf{N}, \mathbf{A}, \mathbf{0}\}$, $P_2 = \{R_1, R_2\}$ where R_1 and R_2 are the following relabelling rules:



with the priority relation: $R_1 >_2 R_2$.

Suppose that (G, λ) is a labelled graph containing exactly one $\mathbf{A}$-labelled vertex. As before, this system computes a spanning tree of G but in a strictly sequential way, using the well-known depth-first search algorithm: the (unique) active vertex, with label $\mathbf{A}$, may activate one of its $\mathbf{N}$-labelled neighbours and become marked

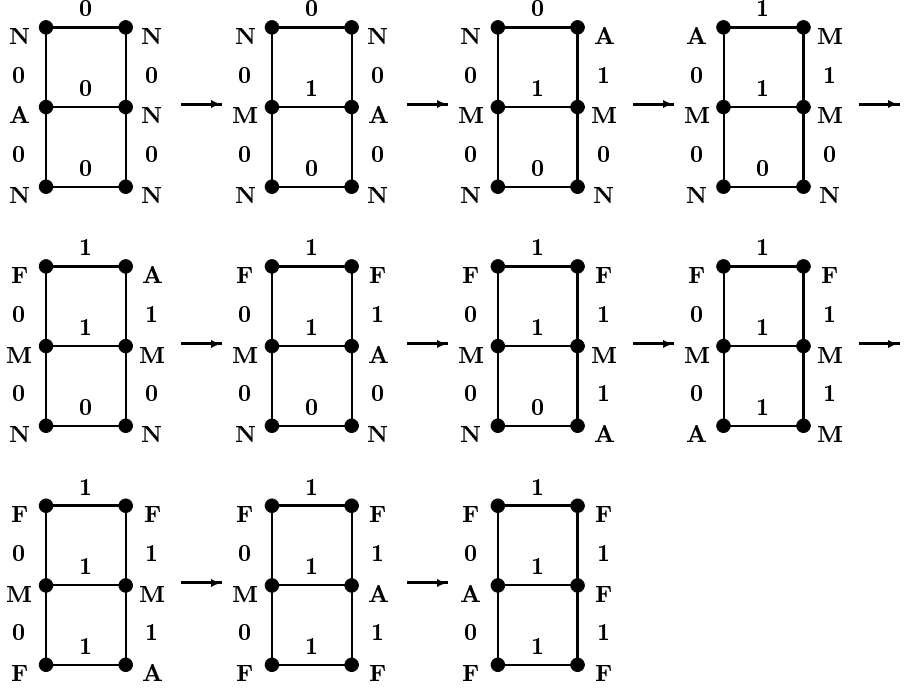
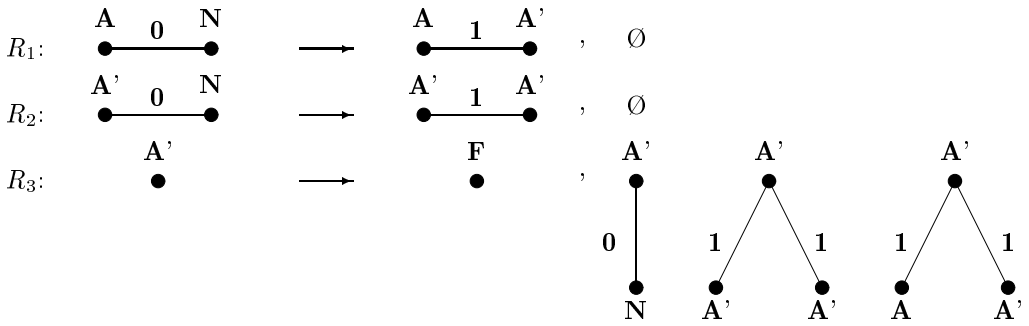


Fig. 2. A sample $\mathcal{R}_2$-relabelling sequence.

(label **M**). When an active vertex has no **N**-labelled neighbour, it reactivates its "father" (which corresponds to the unique **M**-labelled vertex to which it is linked by a **1**-labelled edge), and becomes **F**-labelled. Fig. 2 shows a sample $\mathcal{R}_2$-relabelling sequence.

Let (G, λ) be a labelled graph. A *context* of (G, λ) is a triple (H, μ, ψ) such that (H, μ) is a labelled graph and ψ an occurrence of (G, λ) in (H, μ). A *relabelling rule with forbidden contexts* is a 4-tuple $R = (G_R, \lambda_R, \lambda'_R, F_R)$ such that $(G_R, \lambda_R, \lambda'_R)$ is a relabelling rule and F_R is a finite set of contexts of (G_R, λ_R). A *graph relabelling system with forbidden contexts* (FCGRS for short) is a triple $\mathcal{R} = (\mathcal{L}, \mathcal{I}, P)$ defined as a GRS except that the set P is a set of relabelling rules with forbidden contexts. A *$\mathcal{R}$-relabelling step* is a 5-tuple $(G, \lambda, R, \varphi, \lambda')$ such that R is a relabelling rule with forbidden contexts in P, φ is both an occurrence of (G_R, λ_R) in (G, λ) and an occurrence of (G_R, λ'_R) in (G, λ'), and for every context (H_i, μ_i, ψ_i) of (G_R, λ_R), there is no occurrence φ_i of (H_i, μ_i) in (G, λ) such that $\varphi_i(\psi_i(G_R, \lambda_R)) = \varphi(G_R, \lambda_R)$. In other words, a relabelling rule with forbidden contexts may be applied on some occurrence if and only if this occurrence is not "included" in an occurrence of some of its forbidden contexts.

Example 3. Let $\mathcal{R}_3 = (\mathcal{L}_3, \mathcal{I}_3, P_3)$ be the FCGRS defined by $\mathcal{L}_3 = \{\mathbf{N}, \mathbf{A}, \mathbf{M}, \mathbf{F}, \mathbf{0}, \mathbf{1}\}$, $\mathcal{I}_3 = \{\mathbf{N}, \mathbf{A}, \mathbf{0}\}$, $P_3 = \{R_1, R_2, R_3\}$ where R_1, R_2 and R_3 are the following relabelling rules with forbidden contexts:



The unique vertex of the left-hand side of the rule R_3 is associated with the top vertex of its forbidden contexts. Roughly speaking, the rule R_3 means that a $\mathbf{A}'$-labelled vertex may become **F**-labelled if it has no **N**-labelled neighbour (in that case rule R_2 should be applied) and at most one **A**- or $\mathbf{A}'$-labelled neighbour.

This system provides a distributed implementation of the sequential algorithm encoded in Example 2 (we may have several active vertices, with label **A** or $\mathbf{A}'$, at the same time). Fig. 3 shows a sample $\mathcal{R}_3$-relabelling sequence.

Due to the control mechanism on the applicability of relabelling rules in PGRSs and FCGRSs, only relabelling steps concerning "far enough" occurrences may be applied concurrently. Roughly speaking, in order to check

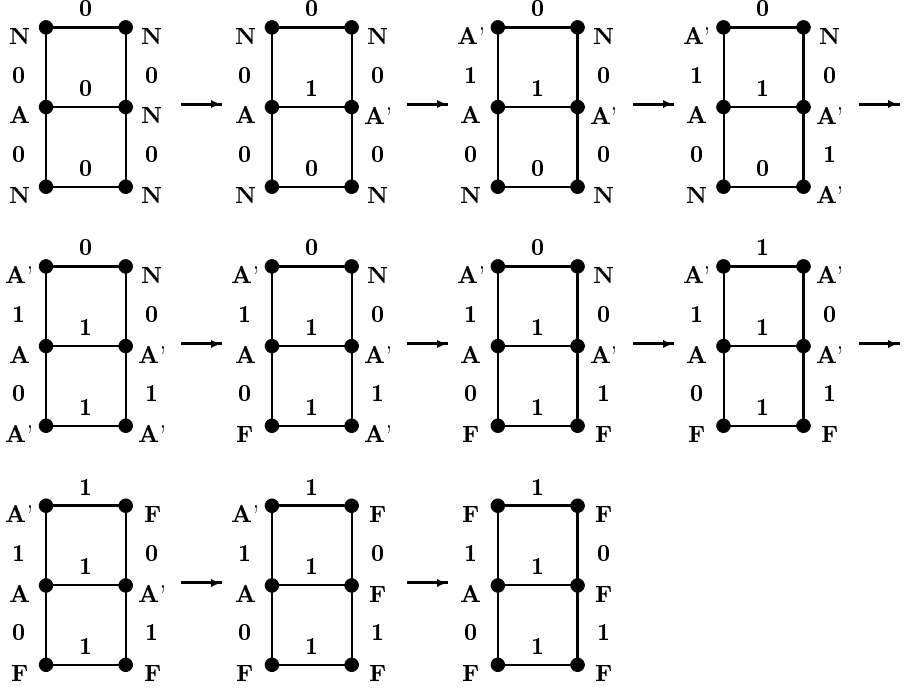


Fig. 3. A sample $\mathcal{R}_3$-relabelling sequence.

whether a relabelling rule may be applied on a given occurrence or not it is necessary to consider some "control area" surrounding this occurrence. Two relabelling steps are then "independant" if their corresponding control areas do not intersect. The reader should note here that the diameter of this control area is bounded by some constant only depending on the graph relabelling system.

The comparison between the expressive power of PGRSs and FCGRSs, together with some other types of GRSs, has been done in [15]. In particular, it has been proved that PGRSs and FCGRSs are equivalent: for every PGRS (resp. FCGRS) there exists a FCGRS (resp. PGRS) achieving the same computation. In the rest of the paper we will thus indifferently provide examples under the PGRS or FCGRS form.

2.3 Local Computations in Graphs

One of the main characteristics of distributed algorithms is the *locality* of the computation. Every computation step occurring on some processor only depends on the local context of this processor. This locality concept is captured via the notion of *local graph relabelling relations* [17].

Let G be a graph, x a vertex in $V(G)$ and k some positive integer. We denote by $B_G(x,k)$ the *ball* of radius k centered at x, that is the subgraph of G induced by all vertices that are at distance at most k from x (recall that the distance between two vertices is the length of a shortest path linking these two vertices). A *graph relabelling relation* (over $\mathcal{L}$) is a binary relation $\mathcal{R}$ defined on the set of $\mathcal{L}$-labelled graphs such that every pair in $\mathcal{R}$ is of the form $((G,\lambda),(G,\lambda'))$. Thus, two labelled graphs in relation only differ on their labelling function. We will write $(G,\lambda)\mathcal{R}(G,\lambda')$ whenever the pair $((G,\lambda),(G,\lambda'))$ is in $\mathcal{R}$. A $\mathcal{L}$-labelled graph (G,λ) is said to be $\mathcal{R}$-*irreducible* if there exists no (G,λ') such that $(G,\lambda)\mathcal{R}(G,\lambda')$. We will denote by $\mathcal{R}^*$ the reflexive and transitive closure of $\mathcal{R}$ and, for every $\mathcal{L}$-labelled graph (G,λ), by $Irred_{\mathcal{R}}(G,\lambda)$ the set of $\mathcal{R}$-irreducible graphs (G,λ') such that $(G,\lambda)\mathcal{R}^*(G,\lambda')$.

We say that a graph relabelling relation $\mathcal{R}$ is k-*local* for some positive integer k if for every pair $((G,\lambda),(G,\lambda'))$ in $\mathcal{R}$, there exists some vertex x in $V(G)$ such that λ and λ' coincide on $V(G)\setminus V(B_G(x,k))\cup E(G)\setminus E(B_G(x,k))$. Intuitively speaking, it means that λ and λ' only differ on a centered ball of radius at most k. A graph relabelling relation is *local* if it is k-local for some k. A graph relabelling relation $\mathcal{R}$ is k-*locally generated* if it can be computed for any graph as soon as it is known on the set of graphs with diameter at most $2k$. More formally, if (G,λ), (G',λ'), (H,μ), (H',μ') are four labelled graphs, $B_G(x,k)$ and $B_H(y,k)$ two isomorphic balls in G and H respectively such that (i) λ and λ' coincide on $V(G)\setminus V(B_G(x,k))\cup E(G)\setminus E(B_G(x,k))$, (ii) μ and μ' coincide on $V(H)\setminus V(B_H(y,k))\cup E(H)\setminus E(B_H(y,k))$ and (iii) λ and μ coincide respectively on $B_G(x,k)$ and $B_H(y,k)$ then $(G,\lambda)\mathcal{R}(G',\lambda')$ if and only if $(H,\mu)\mathcal{R}(H',\mu')$. A graph relabelling relation is *locally generated* if it is k-locally generated for some k.

Graph relabelling systems (GRSs, PGRSs, FCGRSs) are thus special cases of locally generated graph relabelling relations. One of the main questions in that framework is "what can be computed by means of locally

generated graph relabelling relations ?". This question is obviously strongly related to the general problem of characterizing those functions that can be computed by distributed algorithms in an asynchronous way.

3 Proof techniques

Graph relabelling systems provide a formal model for expressing distributed algorithms. The aim of this section is to show that this model is suitable for studying and proving properties of distributed algorithms.

A graph relabelling system $\mathcal{R}$ is *noetherian* if there is no infinite $\mathcal{R}$-relabelling sequence starting from a graph with initial labels in $\mathcal{I}$. Thus, if a distributed algorithm is encoded by a noetherian graph relabelling system then this algorithm always terminates. In order to prove that a given system is noetherian we generally use the following technique. Let $(S, <)$ be a partially ordered set with no infinite decreasing chain (that is every decreasing chain $x_1 > x_2 > \cdots > x_n > \ldots$ in S is finite). We say that $<$ is a *noetherian order compatible with* $\mathcal{R}$ if there exists a mapping f from $\mathcal{G}_{\mathcal{L}}$ to S such that for every $\mathcal{R}$-relabelling step $(G, \lambda, R, \varphi, \lambda')$ we have $f(G, \lambda) > f(G, \lambda')$. It is not difficult to see that if such an order exists then the system $\mathcal{R}$ is noetherian: since there is no infinite decreasing chain in S, there cannot exist any infinite $\mathcal{R}$-relabelling sequence.

In order to prove the correctness of a graph relabelling system, that is the correctness of an algorithm encoded by such a system, it is useful to exhibit (i) some *invariant properties* associated with the system (by invariant property, we mean here some property of the graph labelling that is satisfied by the initial labelling and that is preserved by the application of every relabelling rule) and (ii) some properties of irreducible graphs. These properties generally allow to derive the correctness of the system.

Let us illustrate these techniques by considering the simple graph relabelling system $\mathcal{R}_1$ given in Example 1.
Termination: Let f be the mapping from $\mathcal{G}_{\mathcal{L}_1}$ to the set of natural integers $\mathbb{N}$ which associates with each $\mathcal{L}_1$-labelled graph the number of its **N**-labelled vertices. Observing that this number strictly decreases when we apply the relabelling rule R_1 we get that $(\mathbb{N}, >)$ is a noetherian order compatible with the system $\mathcal{R}_1$. Thus $\mathcal{R}_1$ is a noetherian system.
Correctness: Let (G, λ) be a $\mathcal{L}_1$-labelled graph and P_1, P_2 be the following properties:

P_1 : Every **1**-labelled edge is incident with two **A**-labelled vertices,
P_2 : The subgraph of G made of the **1**-labelled edges and the **A**-labelled vertices has no cycle.

Every $\mathcal{I}_1$-labelled graph satisfies P_1 and P_2 since it has no **1**-labelled edge. Moreover, these two properties are clearly preserved when we apply the rule R_1. Thus, P_1 and P_2 are invariant with respect to $\mathcal{R}_1$.

Let now (G, λ) be any $\mathcal{I}_1$-labelled graph having at least one **A**-labelled vertex and (G, λ') be a labelled graph in $Irred_{\mathcal{R}_1}(G, \lambda)$. Considering the relabelling rule R_1, (G, λ') cannot have any **N**-labelled vertex. From property P_2, we get that the subgraph of (G, λ') induced by the **1**-labelled edges has no cycle. If (G, λ) has exactly one **A**-labelled vertex we thus obtain a spanning tree of G. If (G, λ) has more than one **A**-labelled vertex we obtain a spanning forest having as many components as the number of these initially **A**-labelled vertices.

The reader interested in more substantial examples is referred to [13]. In particular, the graph relabelling systems introduced in Examples 2 and 3 are considered there.

The complexity of a distributed algorithm encoded by a graph relabelling system can also be studied by using classical techniques from rewriting theory. The space complexity is well-captured by the number of labels that are used, and the (sequential) time complexity by the length of a relabelling sequence. The degree of parallelism may also be measured by considering the ratio between the length of a parallel relabelling sequence and the length of a sequential relabelling sequence. Of course, this ratio strongly depends on the specific topology of the graph under consideration.

4 Tools : Coverings and Quasi-coverings

Coverings is a notion known from algebraic topology [21]. They have been used for simulation [3] and for proving impossibility results on distributed computing [1, 8].

Quasi-coverings have been introduced in [24] to obtain impossibility proofs for local detection of global termination. We can note also that the Kronecker product of graphs is useful [1, 6, 4].

4.1 Coverings

We say that a graph G is a *covering* of a graph H if there exists a surjective homomorphism γ from G onto H such that for every vertex v of $V(G)$ the restriction of γ to $N_G(v)$ is a bijection onto $N_H(\gamma(v))$. In particular, $\{\gamma(u), \gamma(v)\} \in E(H)$ implies $\{u, v\} \in E(G)$. The covering is proper if G and H are not isomorphic. It is called connected if G (and thus also H) is connected. A graph G is called *covering-minimal* if every covering from G to some H is a bijection.

By a simple inductive argument we have :

Lemma 1. *Suppose that G is a covering of H via γ. Let T be a subgraph of H. If T is a tree then $\gamma^{-1}(T)$ is a set of disjoint trees, each being isomorphic to T.*

The previous lemma implies :

Lemma 2. *Let H be a connected graph and let G be a covering of H via γ. Then there exists an integer q such that $Card(\gamma^{-1}(v)) = q$, for every $v \in V(H)$.*

Definition 1. *Let G be a covering of H via γ and let q be such that $Card(\gamma^{-1}(v)) = q$ for all $v \in V(H)$. Then the integer q is called the number of* sheets *of the covering. In this case we denote γ as q-*sheeted covering.

The next lemma state a basic property of coverings which is a main argument for the application to local computations.

Lemma 3. *Let G be a covering of H via γ and let $v_1, v_2 \in V(G)$ be such that $v_1 \neq v_2$ and $\gamma(v_1) = \gamma(v_2)$. Then we have $B_G(v_1) \cap B_G(v_2) = \emptyset$.*

Remark 1. If $q = 1$ in Definition 1 then G and H are isomorphic.

We give now a fundamental lemma which establishes the connection between k-coverings and k-locally generated relabelling relations. It states that if G is a k-covering of G' then any $k-$local computation in G' can be lifted to a $k-$local computation in G compatible with the k-covering relation. This is expressed in the following diagram:

$$\begin{array}{ccc} (G,\lambda_1) & \xrightarrow[R^*]{} & (G,\lambda_2) \\ \Big\downarrow \text{k-covering} & & \Big\downarrow \text{k-covering} \\ (G',\lambda_1') & \xrightarrow[R^*]{} & (G',\lambda_2') \end{array}$$

Lemma 4. *Let $\mathcal{R}$ be a k-locally generated relabelling relation and let (G,λ_1) be a k-covering of (G',λ_1') via γ. Moreover, let $(G',\lambda_1')\mathcal{R}^*(G',\lambda_2')$. Then a labelling λ_2 of G exists such that $(G,\lambda_1)\mathcal{R}^*(G,\lambda_2)$ and (G,λ_2) is a k-covering of (G',λ_2').*

4.2 Quasi-coverings.

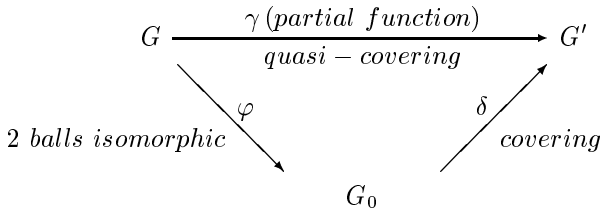
The idea behind the notion of quasi-coverings is to enable the simulation of local computations on a given graph in a restricted area of a larger graph, such that the simulation can lead to false conclusion. The restricted area where we can perform the simulation will shrink while the number of simulated step increases.

Definition 2. *Let G, G' be two graphs and let γ be a partial function on $V(G)$ that assigns to each element of a subset of $V(G)$ exactly one element of $V(G')$.*

Then G is a quasi-covering *of G' of size s if there exists a finite or infinite covering G_0 of G' via δ, vertices $z_0 \in V(G_0)$, $z \in V(G)$, and an integer $r > 0$ such that :*

1. *$B_G(z,r)$ is isomorphic via φ to $B_{G_0}(z_0,r)$,*
2. *$Card(V(B_G(z,r))) \geq s$,*
3. *the domain of definition of γ contains $B_G(z,r))$, and*
4. *$\gamma = \delta \circ \varphi$ when restricted to $V(B_G(z,r))$.*

They correspond to the following diagram :



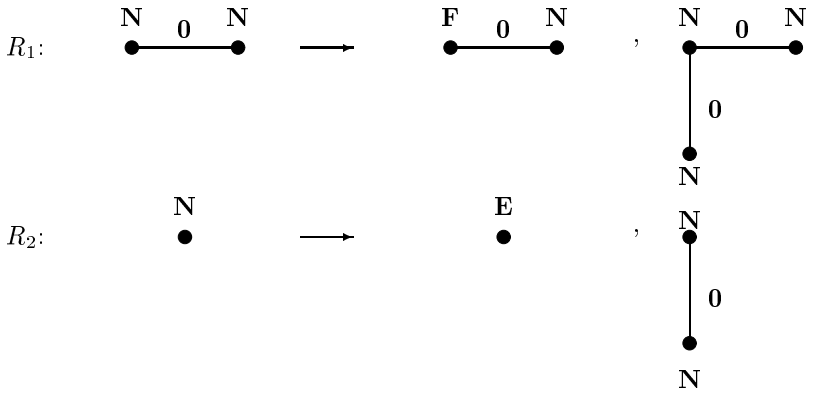
5 The election problem

The election problem is one of the paradigms of the theory of distributed computing [35]. Considering a network of processors we say that a given processor p has been *elected* when the network is in some global state such that the processor p knows that it is the elected processor and all other processors know that they are not. Using our terminology, it means that we get a labelling of the graph in which a unique vertex has some distinguished label.

This problem may be considered under various assumptions [35]: the network may be directed or not, the network may be anonymous (all vertices have the same initial label) or not (every two distinct vertices have distinct initial labels), all vertices, or some of them, may have some specific knowledge on the network or not (such as the diameter of the network, the total number of vertices or simply an upper bound of these parameters), etc.

We first illustrate this problem with a sample FCGRS electing a vertex in a tree.

Example 4. Let $\mathcal{R}_4 = (\mathcal{L}_4, \mathcal{I}_4, P_4)$ be the FCGRS defined by $\mathcal{L}_4 = \{\mathbf{N}, \mathbf{F}, \mathbf{E}, \mathbf{0}\}$, $\mathcal{I}_4 = \{\mathbf{N}, \mathbf{0}\}$ and $P_4 = \{R_1, R_2\}$ where R_1, R_2 are the following relabelling rules with forbidden contexts:



Let us call a *pendant* vertex any **N**-labelled vertex having exactly one **N**-labelled neighbour. The rule R_1 then consists in "cutting" a pendant vertex in the tree, this cut vertex becoming **F**-labelled. Thus, if (G, λ) is a labelled tree whose all vertices are **N**-labelled and all edges are **0**-labelled then this cuting procedure leads to a unique **N**-labelled vertex which becomes elected thanks to the rule R_2.

It is not difficult to observe that every vertex in the tree may be elected by this algorithm. A precise analysis of this algorithm is proposed in [25]. In particular, it is proved that there exist one or two vertices having the highest probability of being elected, namely the *medians* of the graph (recall here that a vertex is called a median if the sum of the distances of this vertex to all other vertices in the graph is minimum).

The more general result we have obtained is the following [9]. We say that v knows the topology of G if v's label encodes the incidence matrix of a graph $G' \simeq G$, but no information enables v to know which vertex of G' corresponds to v. Let G be a graph which is not covering-minimal and let H be such that G is a proper covering of H via the morphism γ. A subgraph K of G is *free modulo* γ if $\gamma^{-1}(\gamma(K))$ is a disjoint union of graphs isomorphic to K. We say that a labelling λ is $\gamma-$*lifted* if

$$\gamma(x) = \gamma(y) \Longrightarrow \lambda(x) = \lambda(y)\,.$$

We say that an algorithm *operates on subgraphs from a given set* S if every relabelling step is performed only on subgraphs of the given graph, which belong to S. Using coverings we prove :

Proposition 1. *Let G be a graph which is not covering-minimal and let γ be a covering from G onto some graph $H \not\simeq G$. Then there is no election algorithm for G which operates on subgraphs free modulo γ, even if the topology of G is known by each vertex of G.*

In [23] Mazurkiewicz gives an election algorithm for the family of graphs which are minimal for the covering relation when we know the size.

The election problem has been considered in the undirected case in [29]. For instance, it has been proved that the election problem can be solved for the so-called *prime graphs*, provided that every vertex knows the total number of vertices in the graph. Let G be an undirected graph and r be a positive integer. A *r-decomposition* of G is a spanning forest of G whose all connected components (trees) contain exactly r vertices. A graph having n vertices is then said to be prime if it only admits 1- and n-decompositions. The class of prime graphs obviously contains all the graphs having a prime number of vertices. We can note that this algorithm enables to present classic ones.

6 The recognition problem

The problem adressed in this section can be informally described as follows: let $\mathcal{F}$ be some class of (unlabelled) graphs. We will say that this class can be *locally recognized* if there exists some graph relabelling system or, more generally, some locally generated graph relabelling relation, which, starting from any uniformly labelled graph (G, λ_0) (that is all vertices and edges have the same label), leads to some final labelling that allows to decide whether G belongs to the class $\mathcal{F}$ or not.

More formally, we define a *graph recognizer* as a pair $(\mathcal{R}, \mathcal{K})$ where $\mathcal{R}$ is a graph relabelling relation and $\mathcal{K}$ a class of labelled graphs. The set of labelled graphs *recognized* by $(\mathcal{R}, \mathcal{K})$ is then defined as the set of labelled graphs (G, λ) such that $Irred_{\mathcal{R}}(G, \lambda) \cap \mathcal{K} \neq \emptyset$. Such a recognizer is said to be *deterministic* if (i) $\mathcal{R}$ is noetherian and (ii) for every labelled graph (G, λ), either $Irred_{\mathcal{R}}(G, \lambda) \subseteq \mathcal{K}$ or $Irred_{\mathcal{R}}(G, \lambda) \cap \mathcal{K} = \emptyset$.

We are essentially interested in graph recognizers where the relation $\mathcal{R}$ is locally generated (with the particular case of graph relabelling systems) and the set $\mathcal{K}$ is defined in some "simple way". In [17] this set $\mathcal{K}$ is defined by means of a so-called *final condition*, that is a logical formula inductively defined as follows: (i) for every label $\ell \in \mathcal{L}$, ℓ is a formula and (ii) if φ and ψ are formulas then so do $\neg\varphi$, $\varphi \vee \psi$ and $\varphi \wedge \psi$. Now, for $\ell \in \mathcal{L}$, a labelled graph satisfies the formula ℓ if it contains at least one ℓ-labelled component, and by induction, it satisfies $\varphi \vee \psi$ if it satisfies φ or ψ and so on in the usual way. Thus, such final conditions allow to verify the presence or the absence of some specific labels but not to count the number of such labels. We will denote by $\mathcal{K}(\varphi)$ the set of labelled graphs which satisfy the formula φ.

Table 1. Recognizable and not-recognizable graph classes

Graph properties	Graphs	1-Graphs
FOL		
exactly one ℓ-labelled vertex	No	Yes
k-regular	Yes	Yes
MSOL		
bipartite	No	Yes
k-colorable ($k > 2$)	No	?
hamiltonian	No	Yes
acyclic	Yes	Yes
SOL		
even number of vertices	No	Yes

In [14, 17] the recognizable classes of graphs are compared to the classes of graphs definable by logic formulas. In particular, it is proved that (deterministically or not) recognizable classes of graphs are not comparable with classes of graphs definable by logic formulas expressed in first-order logic (FOL), monadic second-order logic (MSOL) or second-order logic (SOL). The case of the so-called *1-graphs*, that is graphs having a distinguished vertex is also considered. Table 1 gives some sample graph classes or 1-graph classes that can or cannot be deterministically recognized.

The class of graphs having an even number of vertices can be undeterministically recognized but cannot be deterministically recognized. The class of graphs having an odd number of vertices cannot be recognized, even in an undeterministic way. Thus, the set of deterministically recognizable classes of graphs is not closed under taking complement and is strictly included in the set of undeterministically recognizable classes of graphs. However, the set of deterministically recognizable classes of graphs is closed under union and intersection [17]. We have proved also that classes of graphs defined by forbidden minors are not recognizable in general [7]; in the case of graphs with one source they are recognizable [5].

The main question here is to find some characterization of the classes of graphs that can be recognized by locally generated graph relabelling relations. Up to now, this question is still an open problem. Proof techniques based on coverings have been extended to the case where initial knowledge (e.g. the size) is assumed [9]. A graph G is called covering-minimal if every covering from G to some H is a bijection. The class of covering-minimal graphs plays an important role in the study of local computations. It is easy to verify (using prime rings) that the property of being covering-minimal is not recognizable without any initial knowledge about the graph. Using [23] we note that this property is recognizable if we have as initial knowledge the size of the graph. Having an odd number of vertices or having exactly one vertex with a certain label are other examples of properties which are not recognizable without initial knowledge, but are recognizable if the graph size is known. Thus recognizability under the assumption that the size is known to the algorithm is significantly more powerful than recognizability without initial knowledge.

7 The termination detection problem

Let $\mathcal{R}$ be a locally generated relabelling relation (in this section we assume that $\mathcal{R}$ is a non-constant relation), let (G, λ) a labelled graph, we say that (G, λ) is a terminal configuration modulo $\mathcal{R}$ if (G, λ) is an $\mathcal{R}-$normal form. Let $\mathcal{I}$ be a class of labelled graphs, terminal configurations obtained from $\mathcal{I}$ are said to be locally characterized if there exists a set F of labels such that for any $(G, \lambda) \in \mathcal{I}$ and for any (G, λ'), with $(G, \lambda)\mathcal{R}^*(\mathcal{G}, \lambda')$, (G, λ') is a terminal configuration if and only if there exists a vertex v of (G, λ') having its label in F. In this case termination is said to be explicit. If there exists no sets F of labels which enable the local characterization of terminal configurations, termination is said to be implicit. We study local computations such that terminations are explicit. In [24] it has been proved :

Theorem 1. *Let $\mathcal{I}$ be a class of connected labelled graphs. Suppose that some $\mathbf{G} \in \mathcal{I}$ has connected quasi-coverings in $\mathcal{I}$ of arbitrary large size. Then there is no explicit termination for the class $\mathcal{I}$.*

By combining extended results from [36], the universal reconstruction algorithm of Mazurkiewicz [23] and using the quasi-covering notion we get [28] :

Theorem 2. *Let $\mathcal{I}$ be a class of connected labelled graphs. Suppose that $\forall (G, \lambda) \in \mathcal{I}\ \exists h_{(G,\lambda)} \geq 0$ such that (G, λ) has not quasi-coverings of size greater than $h_{(G,\lambda)}$ in $\mathcal{I}$. Then any locally generated relabelling relation having an implicit termination may be transformed into a locally generated relabelling relation having an explicit termination.*

Well known results are obtained as corollaries : we deduce immediatly that any locally generated relabelling relation having an implicit termination may be transformed into a locally generated relabelling relation having an explicit termination for the following families of graphs :

- graphs having a leader
- graphs such that each node is identified by a unique name
- graphs having a known size or diameter bounds
- trees
- triangulated graphs.

8 Randomized Local Synchronization

Many problems have no solution in distributed computing [18]. The introduction of randomization makes possible tasks that admit no deterministic solutions or simplifies algorithms. General considerations about randomized distributed algorithms may be found in [35] and some techniques used in the design and for the analysis of randomized algorithms are presented in [30, 20, 10].

In [26, 27] we propose and study randomized algorithms to implement distributed algorithms specified by local computations.

More precisely, a star is a complete bipartite graph $K_{1,d}$; the vertex of degree d is called the centre of the star while the other vertices are called the leaves of the star. By K_2 we denote the complete graph of size 2. An algorithm is encoded by means of local relabellings : labels attached to vertices are modified locally, that is on a subgraph isomorphic to a star or to K_2, according to some rules depending on the labels attached to vertices of the star or on the labels attached to the vertices of K_2. We consider three kinds of local computations.

RV : in a computation step, the labels attached to vertices of K_2 are modified according to some rules depending on the labels appearing on K_2.

LC_1 : in a computation step, the label attached to the centre of the star is modified according to some rules depending on the labels of the star, labels of the leaves are not modified.

LC_2 : in a computation step, labels attached to the centre and to the leaves of the star may be modified according to some rules depending on the labels of the star.

In the case of RV two steps may be done in parallel if and only they occur on disjoint edges. In the second case two steps may be done in parallel if and only if they occur on stars having different centres while in the third case two steps may be done in parallel if and only if they occur on disjoint stars.

The aim of local synchronizations is the implementation of such local computations on the following system. We consider an asynchronous distributed network of anonymous processors with an arbitrary topology; processors communicate by exchanging messages. It is represented as a connected graph where vertices represent processors, and two vertices are connected by an edge if the corresponding processors have a direct communication link. We consider systems with asynchronous message passing :

- a process sends a message to another processor by depositing the message in the corresponding channel,

– there is no fixed upper bound on how long it takes for the message to be delivered.

Angluin [1] has proved that there is no deterministic algorithm to implement local synchronizations in an anonymous network that passes messages asynchronously (see [35]). Thus we have no choice but to consider randomized procedures. In this paper, we consider the following distributed randomized procedures.

Implementation of RV. The implementation of RV is the rendezvous. We consider the following distributed randomized procedure. The implementation is partitionned into rounds; in each round each vertex v selects one of its neighbours $c(v)$ at random. There is a rendezvous between v and $c(v)$ if $c(v) = v$, we say that v and $c(v)$ are synchronized. When v and $c(v)$ are synchronized there is an exchange of messages by v and $c(v)$. This exchange allows the two nodes to change their labels. Each message for the synchronization mechanism will be a single bit :

Each vertex v repeats forever the following actions :

the vertex v selects one of its neighbours $c(v)$ chosen at random;
the vertex v sends 1 to $c(v)$;
the vertex v sends 0 to its neighbours different from $c(v)$;
the vertex v receives messages from all its neighbours.
(There is a rendezvous between v and $c(v)$ if v receives 1 from $c(v)$;*
*in this case a computation step may be done. *)*

Randomized Rendezvous

Implementation of LC_1. Let LS_1 the local synchronisation for implementing LC_1; it is partioned into rounds, and in each round, every processor v selects an integer $rand(v)$ randomly from the set $\{1, ..., N\}$. The processor v sends to its neighbours the value $rand(v)$. The vertices of the star centered on v, denoted S_v, are synchronized if for each leave w of S_v : $rand(v) > rand(w)$. In this case a computation step on S_v is allowed : the centre collects labels of the leaves and then changes its label.

Each vertex v repeats forever the following actions :

the vertex v selects an integer $rand(v)$ chosen at random;
the vertex v sends $rand(v)$ to its neighbours;
the vertex v receives integers from all its neighbours.
(There is a LS_1-synchronization centered on v if $rand(v)$ is strictly greater*
than integers received by v; in this case a computation step may be done on
*S_v. *)*

Randomized LS_1-synchronizations.

Implementation of LC_2. Let LS_2 the local synchronisation for implementing LC_2; it is partioned into rounds, and in each round, every processor v selects an integer $rand(v)$ randomly from the set $\{1, ..., N\}$.

The processor v sends to its neighbours the value $rand(v)$. When it has received from each neighbour an integer, it sends to each neighbour w the max of the set of integers it has received from neighbours different from w. The vertex of the star S_v centered on v are synchronized if $rand(v)$ is strictly greater than $rand(w)$ for any vertex w of the ball centered on v of radius 2; In this case a computation step may be done on S_v. During this computation step there is a total exchange of labels by nodes of S_v, this exchange allows nodes of S_v to change their labels.

Each vertex v repeats forever the following actions :

the vertex v selects an integer $rand(v)$ chosen at random;
the vertex v sends $rand(v)$ to its neighbours;
the vertex v receives messages from all its neighbours;
let Int_w the max of the set of integers that v has received from vertices different from w;
the vertex v sends to each neighbour w Int_w;
the vertex v receives integers from all its neighbours;
(There a LS_2-synchronization centered on v if $rand(v)$ is strictly greater*
than all integers received by v; in this case a computation step may be done on
*S_v. *)*

Randomized LS_2-synchronizations.

Our analysis is based on the consideration of *rounds*: in order to measure the performance of the algorithm in terms of the number of synchronization taking place, we assume that at some instant each node sends and receives messages. Thus this parameter of interest, which is the (random) number of synchronizations, is the maximal number (i.e. under the assumption that all nodes are active) authorized by the algorithm. We prove that these three algorithms are Las Vegas algorithms.

References

1. D. Angluin. Local and global properties in networks of processors. In *Proceedings of the 12th Symposium on theory of computing*, pages 82–93, 1980.
2. H. Attiya and J. Welch. *Distributed computing.* McGraw-Hill, 1998.
3. H.-L. Bodlaender and J. van Leeuwen. Simulation of large networks on smaller networks. *Information and Control*, 71:143–180, 1986.
4. A. Bottreau and Y. Métivier. The kronecker product and local computations in graphs. In *Colloquium on trees in algebra and programming*, volume 1059 of *Lecture notes in computer science*, pages 2–16. Spinger-Verlag, 1996.
5. A. Bottreau and Y. Métivier. Minor searching, normal forms of graph relabelling : two applications based on enumerations by graph relabelling. In *Foundations of Software Science and Computation Structures*, volume 1378 of *Lecture notes in computer science*, pages 110–124. Spinger-Verlag, 1998.
6. A. Bottreau and Y. Métivier. Some remarks on the kronecker products of graphs. *Inform. Proc. letters*, 68:55–61, 1998.
7. B. Courcelle and Y. Métivier. Coverings and minors : application to local computations in graphs. *Europ. J. Combinatorics*, 15:127–138, 1994.
8. M. J. Fisher, N. A. Lynch, and M. Merritt. Easy impossibility proofs for distributed consensus problems. *Distrib. Comput.*, 1:26–29, 1986.
9. E. Godard, Y. Métivier, and A. Muscholl. The power of local computations in graphs with initial knowledge. In *Theory and applications of graph transformations*, volume 1764 of *Lecture notes in computer science*, pages 71–84. Spinger-Verlag, 2000.
10. R. Gupta, S. A. Smolka, and S. Bhaskar. On randomization in sequential and distributed algorithms. *ACM Comput. sur.*, 26(1):7–86, 1994.
11. T. Kameda and M. Yamashita. Computing on anonymous networks: Part i - characterizing the solvable cases. *IEEE Transactions on parallel and distributed systems*, 7(1):69–89, 1996.
12. T. Kameda and M. Yamashita. Computing on anonymous networks: Part ii - decision and membership problems. *IEEE Transactions on parallel and distributed systems*, 7(1):90–96, 1996.
13. I. Litovsky and Y. Métivier. Computing trees with graph rewriting systems with priorities. *Tree automata and languages*, pages 115–139, 1992.
14. I. Litovsky and Y. Métivier. Computing with graph rewriting systems with priorities. *Theoret. Comput. Sci.*, 115:191–224, 1993.
15. I. Litovsky, Y. Métivier, and E. Sopena. Different local controls for graph relabelling systems. *Math. Syst. Theory*, 28:41–65, 1995.
16. I. Litovsky, Y. Métivier, and E. Sopena. Graph relabelling systems and distributed algorithms. In H. Ehrig, H.J. Kreowski, U. Montanari, and G. Rozenberg, editors, *Handbook of graph grammars and computing by graph transformation*, volume 3, pages 1–56. World Scientific, 1999.
17. I. Litovsky, Y. Métivier, and W. Zielonka. On the recognition of families of graphs with local computations. *Information and Computation*, 118(1):110–119, 1995.
18. N. Lynch. A hundred impossibility proofs for distributed computing. In *8th International Conference on Distributed Computing Systems*, pages 1–28, 1989.
19. N. A. Lynch. *Distributed algorithms.* Morgan Kaufman, 1996.
20. J. Ramirez-Alfonsin M. Habib, C. McDiarmid and B. Reed, editors. *Probabilistic Methods for Algorithmic Discrete Mathematic.* Springer-Verlag, 1998.
21. W. S. Massey. *A basic course in algebraic topology.* Springer-Verlag, 1991. Graduate texts in mathematics.
22. A. Mazurkiewicz. Trace theory. In W. Brauer et al., editor, *Petri nets, applications and relationship to other models of concurrency*, volume 255 of *Lecture notes in computer science*, pages 279–324. Spinger-Verlag, 1987.
23. A. Mazurkiewicz. Distributed enumeration. *Inf. Processing Letters*, 61:233–239, 1997.
24. Y. Métivier, A. Muscholl, and P.-A. Wacrenier. About the local detection of termination of local computations in graphs. In *International Colloquium on structural information and communication complexity*, pages 188–200, 1997.
25. Y. Métivier and N. Saheb. Probabilistic analysis of an election algorithm in a tree. In *Colloquium on trees in algebra and programming*, volume 787 of *Lecture notes in computer science*, pages 234–245. Spinger-Verlag, 1994.
26. Y. Métivier, N. Saheb, and A. Zemmari. Randomized rendezvous. Technical Report 1228-00, LaBRI, 2000.
27. Y. Métivier, N. Saheb, and A. Zemmari. On the efficiency of randomized local synchronizations. Technical report, LaBRI, en préparation.
28. Y. Métivier and G. Tel. Termination detection and universal graph reconstruction. In *International Colloquium on structural information and communication complexity*, 2000.
29. Y. Métivier and P.-A. Wacrenier. A distributed algorithm for computing a spanning tree in anonymous t-prime graphs. Technical Report 1229-00, LaBRI, 2000.
30. R. Motwani and P. Raghavan. *Randomized algorithms.* Cambridge University Press, 1995.
31. M. Raynal. *Networks and distributed computation : concepts, tools, and algorithms.* MIT Press, Cambridge, 1988.
32. M. Raynal and J.-M Helary. *Synchronization and control of distributed systems and programs.* John Wiley and Sons, Ltd., Chichester,U.K., 1990.
33. K. H. Rosen, editor. *Handbook of discrete and combinatorial mathematics.* CRC Press, 2000.
34. P. Rosenstiehl, J.-R. Fiksel, and A. Holliger. Intelligent graphs. In R. Read, editor, *Graph theory and computing*, pages 219–265. Academic Press (New York), 1972.
35. G. Tel. *Introduction to distributed algorithms.* Cambridge University Press, 1994.
36. B. Szymanski Y. Shi and N. Prywes. Terminating iterative solutions of simultaneous equations in distributed message passing systems. In *4th International Conference on Distributed Computing Systems*, pages 287–292, 1985.