URL: http://www.elsevier.nl/locate/entcs/volume51.html 15 pages

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

M. Bauderon, Y. Métivier, M. Mosbah, A. Sellami

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

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.

Several such models have been defined by: Rosenstiehl et al. [30], Angluin [1] and Yamashita and Kameda [16]. In [30] a synchronous model is considered, where vertices represent (identical) deterministic finite automata. The basic computation step is to compute the next state of each processor according to its state and the states of its neighbours. In [1] an asynchronous model is considered. A basic computation step means that two adjacent vertices exchange their labels and then compute new ones. In [16] an asynchronous model is studied where a basic computation step means that a processor either changes its state and sends a message or it receives a message.

Another common and general approach to concurrent computation is based on *processes* and *actions*. Thereby, a process P_0 can perform some action $A \in \{a_1, \ldots, a_n\}$ or create some new (child) processes $P_1, \ldots, P_m$ in each phase of its life cycle. Such systems are usually described in terms of process algebra or process logic [17], but graph grammars can be used for this purpose as well [15]. This notion of concurrent computing is highly abstract with respect to the underlying hardware, and highly dynamic with respect to the number of processes involved. In our approach, the number of processes (processors) in each model is fixed, but we have a topological notion of the network providing the material basis of distributed computation. In fact, by identifying processes with processors, our notion of distributed computation appears as a useful special case of the actor notion of [15].

Identifying graph nodes with processors and edges between the nodes as communication links between processors is not the only possibility of describing distributed computation in terms of graph transformation systems. In the approach of [28], on the contrary, the communication channels are regarded as nodes and the processes as (hyper) edges. However, the purpose of [28] is to define formal semantics for concurrent programming languages, while our purpose is to provide an experimental platform for observing the runtime behaviour of distributed systems.

The purpose of [31] is similar to ours, namely: tool support for design and examination of distributed software systems. Dealing with brokers, objects and interfaces, their approaches less abstract and closer to software engineering than ours, which is mainly concerned with the abilities of basic distributed algorithms. Also their notion of distributed graph transformation is different: in [31], a distributed graph represents distributed *data* and it is the graph itself whose subgraphs are arbitrarily distributed over the network. In their approach, computation is externally performed *on* the (passive) graph, while our computation is performed *in* the (active) graph. Thus, their computations are user-triggered, while ours are autonomous. Their graphs are highly heterogeneous while ours are highly homogeneous as we have only nodes of one type, all of them running the same internal program.

In our work, 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 [20]. 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.

Then we explain how we obtain an implementation and a visualization of distributed algorithms described by means of local computations into an asynchronous system with asynchronous message passing using randomized algorithms.

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**),

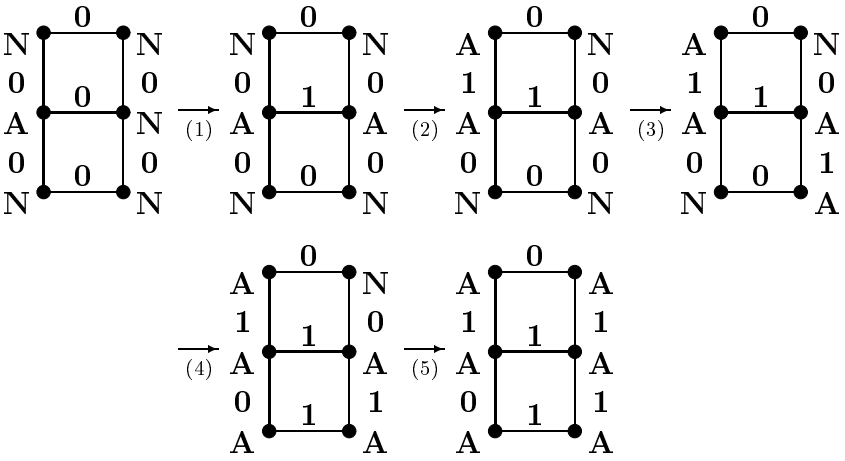

Fig. 1. Computation of a spanning tree.

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.

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 [29]. 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 $\varphi(G, \lambda) = (H', \mu')$.

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. Graph relabelling systems and more generally local computations satisfy the following constraints which seem to be natural when describing distributed computations with a decentralized control:

($C1$) they do not change the underlying graph but only the labelling of its components (edges and/or vertices), the final labelling being the result of the computation,

($C2$) they are *local*, that is, each relabelling step changes only a connected subgraph of a fixed size in the underlying graph,

($C3$) they are *locally generated*, that is, the application condition of the relabelling only depends on the *local context* of the relabelled subgraph.

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) (see [19]) 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.

Coverings and quasi-coverings are fundamental tools to understand the borderline between positive and negative results about distributed algorithms.

Coverings is a notion known from algebraic topology [22]. They have been used for simulation [5] and for proving impossibility results on distributed computing [1,11]. 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 [1,8,6].

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.

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 (see [24]).

4 Some Classical Problems

Among paradigms associated with local computations, we can cite the spanning tree computation, the termination detection problem, the election problem and the recognition problem.

Spanning Tree Computation

Computing a spanning tree is standard in the domain of distributed algorithms. Trees are an essential structure in various communication protocols (synchronization, deadlock resolution, information broadcasting). This problem is closely related to the election problem (see below). Depth-first search trees are used for the construction of interval labeling schemes for compact routing. Several examples of algorithms for computing spanning trees encoded by local computations are given in [20].

The Termination Detection Problem

In an implicitly terminating algorithm, each execution is finite and in the last state of the execution each node has the correct result. However, the nodes are not aware that their state is the last one in the execution. Termination is said to be explicit in a process if that process is in a terminal configuration and its state is terminal. There were many proposals for *termination detection* algorithms: such algorithms transform implicitly into explicitly terminating algorithms. Several conditions were found to allow such algorithms and for each of these conditions a specific algorithm was given (see [32]). These conditions include: - A unique *leader* exists in the network [1], - The network is known to be a tree [1], - The diameter of the network is known [33], - The nodes have different identification numbers. These four conditions are just special cases of one common criteria, namely that the local knowledge of nodes prohibits the existance of quasi-coverings of unbounded depth: it is the result presented in this part.

First we need some notation. 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 [27] it has been proved :

Theorem 4.1 *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.

The Election Problem

The election problem is one of the paradigms of the theory of distributed computing [32]. 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 [32]: 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. A general impossibility result which summarize previous results has been obtained in [13]. In [23] Mazurkiewicz gives an election algorithm for the family of graphs which are minimal for the covering relation when we know the size; by combining this result and the termination detection characterization we obtain a characterization of families of graphs for which there exists an election algorithm:

Theorem 4.2 *Let $\mathcal{I}$ be a class of connected labelled graphs. There exists an election algorithm for $\mathcal{I}$ if and only if elements of $\mathcal{I}$ are minimal for the covering relation and $\forall (G,\lambda) \in \mathcal{I}\ \exists h_{(G,\lambda)} \geq 0$ such that (G,λ) has not quasi-coverings of size greater than $h_{(G,\lambda)}$ in $\mathcal{I}$.*

The Recognition Problem

Let L be a set of labels. The problem addressed in this section can be informally described as follows. Let $\mathcal{F}$ be some class of (labelled) graphs. We will say that $\mathcal{F}$ can be *locally recognized* if there exists some graph relabelling system (or, more generally, some locally generated graph relabelling relation) such that starting from any uniformly labelled graph (G,λ_0) some final labelling can be reached that allows to decide whether G belongs to the class $\mathcal{F}$ or not.

Several basic properties like regularity, completeness, or acyclicity can be recognized by local computations without initial knowledge. On the other hand, we cannot determine whether a graph is planar by local computations, provided that the given graph is labelled in an uniform way without initial knowledge [21,9]. However, the presence of a distinguished vertex allows to gather information. In particular, it has been shown that it is possible to detect a given minor in a graph with a distinguished vertex [7], hence also to determine whether the graph is planar. A natural question is whether some additional information encoded in the initial uniform labelling of a graph can

help for deciding for example planarity. Results and main definitions are from [12,21]. A *labelled graph recognizer* is a pair $(\mathcal{R}, \mathcal{K})$ where $\mathcal{R}$ is a graph relabelling relation and $\mathcal{K}$ is a class of labelled graphs. A graph (G, λ) is *recognized* if $\text{Irred}_{\mathcal{R}}(G, \lambda) \cap \mathcal{K} \neq \emptyset$.

We are interested in recognizing graphs which have a certain initial knowledge encoded in the initial labelling. Let G be a graph and α a label in L. Then Λ_α is the uniform labelling on G with α, that is every vertex is labelled by α.

Definition 4.3 A *graph recognizer with initial knowledge* is a triple $(\mathcal{R}, \mathcal{K}, \iota)$ where $(\mathcal{R}, \mathcal{K})$ is a labelled graph recognizer, and ι is a function which associates with each graph G a label $\iota(G) \in L$. The set of graphs *recognized* by $(\mathcal{R}, \mathcal{K}, \iota)$ is given as $\{G \mid (G, \Lambda_{\iota(G)}) \text{ is recognized by } (\mathcal{R}, \mathcal{K})\}$.

A recognizer $(\mathcal{R}, \mathcal{K}, \iota)$ is said to be *deterministic* if, restricted to inputs $(G, \Lambda_{\iota(G)})$, we have the following two properties:

- $\mathcal{R}$ is noetherian.
- Either $\text{Irred}(G, \Lambda_{\iota(G)}) \cap \mathcal{K} = \emptyset$ or $\text{Irred}(G, \Lambda_{\iota(G)}) \subseteq \mathcal{K}$.

We can now define recognizable classes of graphs. A class $\mathcal{F}$ of graphs is said to be *(deterministically) recognizable with initial knowledge* ι if there exists a locally generated (deterministic) graph recognizer $(\mathcal{R}, \mathcal{K}, \iota)$ recognizing exactly $\mathcal{F}$.

We define the relation σ^ι by letting $G \ \sigma^\iota \ G'$ if: $\iota(G) = \iota(G')$ and there exists a graph H such that G and G' are coverings of H. Let $\sim^\iota$ denote the reflexive, transitive closure of σ^ι. A class of graphs $\mathcal{F}$ will be said to be closed under $\sim^\iota$ if for any graphs G and G' such that $G\sigma^\iota G'$, G is in $\mathcal{F}$ if and only if G' is in $\mathcal{F}$. The following useful characterization has been obtained [12]:

Theorem 4.4 *Let $\mathcal{F}$ be a class of graphs and ι an initial knowledge. The following statements are equivalent.*

(i) *$\mathcal{F}$ is locally recognizable with initial knowledge ι.*

(ii) *$\mathcal{F}$ is closed under $\sim^\iota$ and $\mathcal{F}$ is a recursive set.*

Comparison with Logical Languages

In [18,21] 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.

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

5 Implementation

5.1 Randomized Local Elections

In [25,26] we propose and study randomized algorithms to implement distributed algorithms specified by local computations. We recall that 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; K_2 is the complete graph with 2 vertices. 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.

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 [32]). 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 partitioned 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 LE_1 the local election 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 vertex v is elected in the star centered on d and denoted S_v, 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.
(The vertex v is elected if $rand(v)$ is strictly greater than integers received by v; in this case a computation step may be done on S_v. *)*

Randomized LE_1-Elections.

Implementation of LC_2. Let LE_2 the local election 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 v center of the star S_v is elected 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 is a LE_2-election of 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 LE_2-elections.

It has been proved in [25,26] that these three algorithms are Las Vegas algorithms. In the following, we will introduce a tool that we have developed to program the previous algorithms.

5.2 ViSiDiA: a Tool for the Visualization and Simulation of Distributed Algorithms

ViSiDiA [3,2] has been developed in order to help in the design, the experimentation, the validation and the visualization of distributed algorithms discribed by relabelling systems. It allows the user to model a network, to implement and to execute a relabelling system.

- Network construction:
  A friendly Graphical User Interface to draw a graph which will model the network. The user can add, delete, or select vertices, edges or subgraphs. Visual attributes of vertices and edges such as labels, colors or shapes have default values, but they can be easily customized, for instance to assign an initial labelling to the vertices of the graph, such as a label A to a particular vertex, and N to all other vertices.
- Relabelling system implementation:
  The tool provides a library of high level primitives to program the corresponding local computations[3]. The java code given in algorithm 1 shows the implementation of the spanning tree example discussed in Section 2.

  Detailed implementations and descriptions of these algorithms can be found in [4,3]. Note that one has to choose first the type of local computation by choosing one of the three primitives: rendezVous(), starSynchro1(),

Algorithm 1 Implementation of SPANNING-TREE

```
while (run) {
   neighbour = rendezVous();
   sendTo(neighbour,myLabel);
   neighbourLabel=receiveFrom(neighbour);
   if (myLabel == 'N') && (neighbourLabel == 'A'){
        myLabel = 'A';
        edge[neighbour]=1
    }
   breakSynchro();
}
```

and starSynchro2()), which correspond respectively to RV, LE1, and LE2 introduced above. More precisely, these primitives are programmed as follows:

- rendezVous(): a function that returns the neighbour with whom the synchronization occurs.
- starSyncho1(): returns the center of the star during a star synchronization. Only the center can update its attributes.
- starSynchro2(): returns the center of the star during a star synchronization. The center and its neighbours can update their attributes.

- Execution:
  After compiling the module implementing the relabelling system, the user can execute it by pressing on appropriate buttons provided by the interface. The system automatically creates and assigns to each vertex a java thread which will run a copy (a clone) of the code implementing the relabelling system. The user can observe the messages exchanged between vertices (threads), and their states. In particular, label changes of vertices can be seen on-line. The whole algorithm is animated in such a way that the user can follow its execution. Moreover, the number of exchanged messages is computed and displayed. This can be used to perform experiments on particular distributed algorithms described within our framework.

Many distributed algorithms described by relabelling systems are already implemented and can be directly animated[4]. These include the following :

- leader election in trees, in chordal graphs and in complete graphs,
- randomized Rendez-vous and randomized local elections,
- spanning tree in anonymous networks,
- spanning tree in non anonymous networks,
- Mazurkiewicz's universal graph reconstruction,
- Detection of stable properties.

An interesting advantage of our approach is that we only need to implement local rewritings to code complicated distributed algorithms. Therefore, visual-

izing the execution of these algorithms consists in animating distributed local computations.
As an example, let us mention that an implementation of Mazurkiewicz's algorithm based on Estelle specification has been given in [10]. Our tool provides a much easier and almost automatic way to implement it [4], as it can be described within the general framework described in this paper.

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] M. Bauderon, S. Gruner, Y. Métivier, M. Mosbah, and A. Sellami. Visualization of distributed algorithms based on labeled rewriting systems. In *Second International Workshop on Graph Transformation and Visual Modeling Techniques, Crete, Greece*, July 12-13, 2001.

[3] M. Bauderon, S. Gruner, and M. Mosbah. A new tool for the simulation and visualization of distributed algorithms. Technical Report 1245-00, LaBRI, 2000. Accepted in MFI'01, Toulouse, 21-23 May 2001.

[4] M. Bauderon, Y. Métivier, M. Mosbah, and A. Sellami. From local computations to asynchronous message passing systems. Technical report, LaBRI, 2001. In preparation.

[5] H.-L. Bodlaender and J. van Leeuwen. Simulation of large networks on smaller networks. *Information and Control*, 71:143–180, 1986.

[6] 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.

[7] 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.

[8] A. Bottreau and Y. Métivier. Some remarks on the kronecker products of graphs. *Inform. Proc. letters*, 68:55–61, 1998.

[9] B. Courcelle and Y. Métivier. Coverings and minors : application to local computations in graphs. *Europ. J. Combinatorics*, 15:127–138, 1994.

[10] P. Dembinski. Enumeration protocol in estelle: an exercise in stepwise development. In A. Cavalli S. Budkowski and E. Najm, editors, *Formal description techniques and protocol specification, testing and verification*, pages 147–161. Kluwer Academic Publishers, 1998.

[11] M. J. Fisher, N. A. Lynch, and M. Merritt. Easy impossibility proofs for distributed consensus problems. *Distrib. Comput.*, 1:26–29, 1986.

[12] E. Godard and Y. Métivier. A characterization of classes of graphs recognizable by local computations with initial knowledge. *Technical Report, LaBRI*, 2001. Accepted In International Colloquium on structural information and communication complexity 2001.

[13] 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.

[14] H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 3: Concurrency, Parallelism, and Distribution.* World Scientific, 1999.

[15] D. Janssens. Actor grammars and local actions. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *[14]*, chapter 2, pages 57–106.

[16] 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.

[17] M. Leuschel. Design and implementation of the high-level specification language csp(lp). In *Proceedings of PADL'01, 3rd International Symposium on Practical Aspects of Declarative Languages*, Lecture notes in computer science, pages 14–28. Springer-Verlag, October 2001.

[18] I. Litovsky and Y. Métivier. Computing with graph rewriting systems with priorities. *Theoret. Comput. Sci.*, 115:191–224, 1993.

[19] I. Litovsky, Y. Métivier, and E. Sopena. Different local controls for graph relabelling systems. *Math. Syst. Theory*, 28:41–65, 1995.

[20] 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.

[21] 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.

[22] W. S. Massey. *A basic course in algebraic topology.* Springer-Verlag, 1991. Graduate texts in mathematics.

[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, N. Saheb, and A. Zemmari. Randomized rendezvous. In *Mathematics and computer science : Algorithms, trees, combinatorics and probabilities*, Trends in mathematics, pages 183–194, 2000.

[26] Y. Métivier, N. Saheb, and A. Zemmari. Randomized local elections : probabilistic and efficiency analysis. Technical report, LaBRI, en préparation.

[27] Y. Métivier and G. Tel. Termination detection and universal graph reconstruction. In *International Colloquium on structural information and communication complexity*, pages 237–251, 2000.

[28] U. Montanari, M. Pistore, and F. Rossi. In *[14]*, chapter 4, pages 189–268.

[29] K. H. Rosen, editor. *Handbook of discrete and combinatorial mathematics.* CRC Press, 2000.

[30] 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.

[31] G. Taentzer, M. Koch, I. Fischer, and V. Volle. Distributed graph transformation with application to visual design of distributed systems. In *[14]*, chapter 5, pages 269–340.

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

[33] 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.