Termination Detection of Local Computations

Emmanuel Godard, Yves Métivier, Gerard Tel

Termination Detection of Local Computations

Emmanuel Godard[1], Yves Metivier[2] and Gerard Tel[3]

[1] emmanuel.godard@lif.univ-mrs.fr
Laboratoire d'Informatique Fondamentale
CNRS (UMR 6166) – Universite de Provence
39 rue Joliot-Curie
13453, Marseille Cedex 13, France
[2] metivier@labri.fr
LaBRI
Universite Bordeaux 1, ENSEIRB,
351 cours de la Liberation
33405 Talence, France
[3] gerard@cs.uu.nl
Department of Computer Science
University of Utrecht
P.O. Box 80.089, 3508 TB Utrecht
The Netherlands

Abstract. Contrary to the sequential world, the processes involved in a distributed system do not necessarily know when a computation is globally finished. This paper investigates the problem of the detection of the termination of local computations.
We define four types of termination detection: no detection, detection of the local termination, detection by a distributed observer, detection of the global termination. We give a complete characterisation (except in the local termination detection case where a partial one is given) for each of this termination detection and show that they define a strict hierarchy. These results emphasise the difference between computability of a distributed task and termination detection.
Furthermore, these characterisations encompass all standard criteria that are usually formulated : topological restriction (tree, rings, or triangulated networks ...), topological knowledge (size, diameter ...), and local knowledge to distinguish nodes (identities, sense of direction). These results are now presented as corollaries of generalising theorems. As a very special and important case, the techniques are also applied to the election problem. Though given in the model of local computations, these results can give qualitative insight for similar results in other standard models.
The necessary conditions involve graphs covering and quasi-covering; the sufficient conditions (constructive local computations) are based upon an enumeration algorithm of Mazurkiewicz and a stable properties detection algorithm of Szymanski, Shi and Prywes.

Table of Contents

1 Introduction

This paper presents results concerning two fundamental problems in the area of distributed computing: the termination detection problem and the election problem. The proofs are done in the model of local computations and use mainly common results and tools. Namely, they use Mazurkiewicz' algorithm [Maz97], the Szymanski-Shi-Prywes algorithm [SSP85], coverings and quasi-coverings of graphs.

1.1 The Model

We consider networks of processors with arbitrary topology. A network is represented as a connected, undirected graph where vertices denote processors and edges denote direct communication links. Labels are attached to vertices and edges. The identities of the vertices, a distinguished vertex, the number of processors, the diameter of the graph or the topology are examples of labels attached to vertices; weights, marks for encoding a spanning tree or the sense of direction are examples of labels attached to edges.

The basic computation step is to modify labels locally, that is, on a subgraph of fixed radius 1 of the given graph, according to certain rules depending on the subgraph only (local computations). The relabelling is performed until no more transformation is possible, i.e., until a normal form is obtained. This is a model first proposed by A. Mazurkiewicz [Maz88].

This model has numerous interests. As any rigorously defined model, it gives an abstract tool to think about some problems in the field of distributed computing independently of the wide variety of models used to represent distributed systems [LL90]. As classical models in programming, it enables to build and to prove complex systems, and so, to get them right. And quoting D. Angluin in [Ang80], this kind of model makes it possible to put forward phenomena common to other models. It is true that this model is strictly stronger than other standard models (like message passing systems), but then, impossibility results remains true in weaker models. Furthermore, any positive solution in this model may guide the research of a solution in a weaker model or be implemented in a weaker model using randomised algorithms. Finally, this model gives nice properties and examples using classical combinatorial material, hence we believe this model has a very light overhead in order to understand and to explain distributed problems.

We acknowledge, and underline, that the results presented here might be quantitatively different from other models, but we claim that they are not significantly different: they are qualitatively similar, as are all the impossibility results proved in different models since the seminal work of Angluin. All of them use the same "lifting technique", even though not on exactly the same kind of graph morphism [Ang80,Maz97,YK96,BV02c]. Thus it seems possible to extend the general results of this paper to more standard models like the "message passing model". Moreover, this direction has already given some results [CGMT07,CM07,CGM08]. Note also that all the questions addressed in

this paper are not specific of the model of local computations. E.g, is there a unique (universal) algorithm that can solve the election problem on the family G_{min} of networks that admit an election algorithm? Though this very set G_{min} can be different depending on the model of computations that is used, we claim that the generic answer is no and that our main impossibility result can be extended to any other model. The reader should note that this question has not been previously thoroughly answered in any model (see the discussion about the election problem on Section 8.3).

1.2 Related Works

Among models related to our model there are local computation systems as defined by Rosenstiehl et al. [RFH72], Angluin [Ang80], Yamashita and Kameda [KY96], Boldi and Vigna [BV99,BV01] and Naor and Stockmeyer [NS95]. In [RFH72] 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 [Ang80] an asynchronous model is considered. A basic computation step means that two adjacent vertices exchange their labels and then compute new ones. In [KY96] 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. In [BV99,BV01] networks are directed graphs coloured on their arcs; each processor changes its state depending on its previous state and on the states of its in-neighbours. Activation of processors may be synchronous, asynchronous or interleaved. In [NS95] the aim is a study of distributed computations that can be done in a network within a time independent of the size of the network.

1.3 The Termination Detection Problem

Starting with the works by Angluin [Ang80] and Itai and Rodeh [IR81], many papers have discussed the question: what functions can be computed by distributed algorithms in networks where knowledge about the network topology is limited?

Two important factors limiting the computational power of distributed systems are symmetry and explicit termination. Some functions can be computed by an algorithm that terminates implicitly but not by an explicitly terminating algorithm. 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; with an explicitly terminating algorithm, nodes know the local or global termination of the algorithm.

Known Results about the Termination Detection Problem. Impossibility proofs for distributed computations quite often use the replay technique. Starting from a (supposedly correct) execution of an algorithm, an execution is

constructed in which the same steps are taken by nodes in a different network. The mechanics of distributed execution dictate that this can happen, if the nodes are locally in the same situation, and this is precisely what is expressed by the existence of coverings. The impossibility result implies that such awareness can never be obtained in a finite computation. During the nineteen eighties there were many proposals for termination detection algorithms: such algorithms transform implicitly into explicitly terminating algorithms. Several conditions were found to allow such algorithms (thus to nullify the difference between implicitly and explicitly computable functions) and for each of these conditions a specific algorithm was given (see [Mat87,Lyn96,Tel00]). These conditions include:

1. a unique leader exists in the network,
2. the network is known to be a tree,
3. the diameter of the network is known,
4. the nodes have different identification numbers.

The Main Result. In this paper we show that these four conditions are just special cases of one common criterion, namely that the local knowledge of nodes prohibits the existence of quasi-coverings of unbounded radius. We also prove, by generalising the existing impossibility proofs to the limit, that in families with quasi-coverings of unbounded radius, termination detection is impossible. Informally, we prove (see Theorem 6.11):

A distributed task $T = (F, S)$ is locally computable with explicit termination detection if and only if

1.0.i S is covering-lifting closed on F,
1.0.ii there exists a recursive function r such that for any H, there is no strict quasi-covering of H of radius $r(H)$ in F.

Actually, we investigate different termination detection schemes: local termination detection, observed termination detection and global termination detection. This is explained later in this introduction.

This is the first time, to our knowledge, that computability of a distributed task (that is known to relate to "local symmetries") is fully distinguished from the problem of detecting a kind of termination of a distributed computation.

Structural Knowledge and Labelled Graphs The definition of coverings and quasi-coverings are extended to include node and link labellings as well. In the extension it is required that a node is mapped to a node with the same label, and links are mapped to links with the same label. Our approach then naturally abstracts away the difference between anonymous or non-anonymous, centred or uniform networks. Indeed, the network being centred is modelled by considering as local knowledge that the graph family is the collection of graphs that contain exactly one node with the label leader.

Specific assumptions (leader, identities, sense of direction, knowledge of size) now are examples of local knowledge that prevents certain quasi-coverings, thus

allowing termination detection to take place. Weak sense of direction (WSD) allows to distinguish closed from open walks, which is sufficiently strong to rule out all non-trivial quasi-coverings. Thus termination detection is possible in all systems with WSD.

1.4 The Election Problem

As a very fundamental and illustrative problem, we investigate the election problem. The election problem is one of the paradigms of the theory of distributed computing. It was first posed by LeLann [LeL77]. Considering a network of processors the election problem is to arrive at a configuration where exactly one processor is in the state elected and all other processors are in the state non-elected. The elected vertex is used to make decisions, to centralise or to broadcast some information.

Known Results about the Election Problem. Graphs where election is possible were already studied but the algorithms usually involved some particular knowledge. Solving the problem for different knowledge has been investigated for some particular cases (see [AW04,Lyn96,Tel00] for details) including:

1. the network is known to be a tree,
2. the network is known to be complete,
3. the network is known to be a grid or a torus,
4. the nodes have different identification numbers,
5. the network is known to be a ring and has a known prime number of vertices.

The classical proof techniques used for showing the non-existence of election algorithm are based on coverings [Ang80], which is a notion known from algebraic topology [Mas91]. A graph G is a covering of a graph H if there is a surjective morphism from G to H which is locally bijective. The general idea, used for impossibility proofs, is as follows. If G and H are two graphs such that G covers H and $G \not\simeq H$, then every local computation on H induces a local computation on G and every label which appears in H appears at least twice in G: Thus using H it is always possible to build a computation in G such that the label elected appears twice. By this way it is proved that there is no election algorithm for G and H ([Ang80] Theorem 4.5).

A labelling is said to be locally bijective if vertices with the same label are not in the same ball and have isomorphic labelled neighbourhoods. A graph G is non-ambiguous if any locally bijective labelling is bijective. Mazurkiewicz has proved that, knowing the size of graphs, there exists an election algorithm for the class of non-ambiguous graphs [Maz97]. This distributed algorithm, applied to a graph of size n; assigns bijectively numbers of $[1:n]$ to vertices of G: The elected vertex is the vertex having the number 1:

In [MMW97] the notion of quasi-covering has been introduced to study the problem of termination detection. A graph G is a quasi-covering of a graph H if G is locally a covering of H (locally means that there is a vertex v of G and a positive integer k such that the ball centred on v of radius k is a covering of a ball of H).

The Main Result. We characterise which knowledge is necessary and sufficient to have an election algorithm, or equivalently, what is the general condition for a class of graphs to admit an election algorithm: Theorem 8.5. Sufficient conditions given below are just special cases of criteria of Theorem 8.5.

We explain new parts in this theorem. It is well known (see above) that the existence of an election algorithm needs graphs minimal for the covering relation. We prove in this paper that if a graph is minimal for the covering relation and admits quasi-coverings of arbitrary large radius in the family there is no election algorithm. This part can be illustrated by the family of prime rings. Indeed, prime rings are minimal for the covering relation nevertheless there is no election algorithm for this family: without the knowledge of the size, a ring admits quasi-covering prime rings of arbitrary large radius.

These two results prove one sense of Theorem 8.5. To prove the converse:

- We remark that non-ambiguous graphs are precisely graphs which are minimal for the covering relation.
- We extend the Mazurkiewicz algorithm to labelled graphs.
- We prove that the Mazurkiewicz algorithm applied in a labelled graph G enables the "cartography", on each node of G, of a labelled graph H such that G is a quasi-covering of H; and when the computation is terminated G is a covering of H:
- We define and we use an extension of an algorithm by Szymanski, Shi and Prywes [SSP85] which enables the distributed detection of stable properties in a labelled graph.
- We prove that the boundedness of the radius of quasi-coverings of a given labelled graph enables to each node v to detect the termination of the Mazurkiewicz algorithm and finally each node can decide if it is elected by testing if it has obtained number 1 by the Mazurkiewicz algorithm.

1.5 Tools

Coverings, Computations and Symmetry Breaking. The first step of a node in a distributed computation depends only on local initial knowledge of this node; only after receiving information from neighbours, the steps may depend on initial knowledge of these neighbours. (Here initial knowledge includes the node's input, topological knowledge, degree, etc.) Thus, consider a labelled graph G that contains a node v with initial knowledge x, executing a distributed algorithm A. If G contains another node, w say, with the same initial knowledge, or a different labelled graph H contains a node with this knowledge, these nodes may thus execute the same first step if A is executed. Now let v in G have neighbours with initial knowledge a, b, and c and assume that in the labelled graph H, node w also has neighbours with initial knowledge a, b, and c. We thus create a "local similarity" to G of, in this case, a radius 1. In this situation, not only will node w start with the same step as node v, but also will receive the same information after the first step, and consequently will also perform the same second step.

Distributed tasks like election, enumeration (assigning different numbers to the nodes), and mutual exclusion require the network to reach a non-symmetric state. A network state is symmetric if it contains different nodes that are in exactly the same situation; not only their local states, but also the states of their neighbours, of their neighbours' neighbours, etc. That is, there exists a "local similarity" between different nodes of infinite radius.

The replay argument shows that different nodes that are locally similar within finite radius will exhibit the same behaviour in some finite computation. Thus, there is no algorithm that guarantees that the symmetry ceases in all finite computations. Symmetry could be broken only by randomised protocols.

It is not difficult to see that local similarity of infinite radius may exist in finite graphs. The classical example is a ring R_6 of six nodes, with initial states a, b, c, a, b, c. Indeed, the two nodes with state a both have neighbours in state b and c, and so on, so the local similarity exists over an infinite radius.

The ring R_6 can be mapped into a ring R_3 with only three nodes, with initial states a, b, and c, in such a way that each node is mapped to a node with the same initial state and with the same states in neighbours. Such a mapping is called a covering and is the mathematical tool to prove the existence of symmetries.

The Mazurkiewicz Algorithm. The proofs of our results used the fundamental Mazurkiewicz distributed enumeration algorithm. A distributed enumeration algorithm on a graph G is a distributed algorithm such that the result of any computation is a labelling of the vertices that is a bijection from $V(G)$ to $\{1,2,\ldots,|V(G)|\}$. In [Maz97], Mazurkiewicz presents a distributed enumeration algorithm for the class of non-ambiguous graphs (graphs such that any local bijective labelling is a bijective labelling). In this paper we prove that the family of non-ambiguous graphs is the family of graphs minimal for the covering relation.

We prove also that a run of the Mazurkiewicz algorithm on a labelled graph G (not necessarily minimal for the covering relation) enables the computation on each vertex of G of a graph H quasi-covered by G (the quasi-covering becomes a covering when the algorithm halts): we obtain a universal algorithm.

The Szymanski, Shy and Prywes Algorithm and Quasi-Coverings Relate to Termination Detection. Termination detection requires that a node certifies, in a finite computation, that all nodes of the network have completed their computation. However, in a finite computation only information about a bounded region in the network can be gathered. The algorithm by Szymanski, Shy, and Prywes does this for a region of pre-specified diameter; the assumption is necessary that the diameter of the entire network is known. This implies that, termination detection, unlike symmetry breaking, is possible in every graph, but provided some knowledge.

Network knowledge in an algorithm is modelled by a graph family in which the algorithm is required to work. The detection algorithm by Szymanski et al. can be generalised in this way to work in a labelled graph family F. Nodes observe their neighbourhood and determine in what labelled graph H of F they are.

Then they try to get a bound k on the radius to which a different labelled graph of F can be locally similar to H, and then certify that all nodes within distance k are completed. The universal termination detection algorithm thus combines the Mazurkiewicz algorithm with (minimal) topological knowledge [Maz97] and a known termination detection algorithm.

Of course the approach fails if a labelled graph $H \in F$ is locally similar, with unbounded radius, to other graphs in F. Local similarities of this type are made precise in the notion of quasi-coverings. Fortunately, the impossibility proofs for termination detection can be extended to cover exactly those families of labelled graphs that contain such unbounded-radius coverings. Consequently, the sketched universal termination detection algorithm is the most general algorithm possible.

Other Termination Detections In fact, in the previous algorithm, what is detected is that all output values are correctly computed: the task is terminated, the distributed algorithm is not terminated. Indeed, without symmetry breaking conditions, we cannot detect the end of the algorithm. Given a symmetric network, the "last" step can be performed on at least two nodes. We call this kind of detection observed termination detection because in this case, the algorithm acts as an "observer" that knows when the underlying computation of values is finished. We do not ask this observer algorithm to detect its own termination. Thus we distinguished the detection of the global termination of the task from the detection of the termination of the detection... This is presented in Theorem 6.11.

In order to precise what can be explicit termination, we define also other kinds of termination detection: detection of the local termination (the nodes know when they enter their final step) and global termination detection (one node knows that the distributed algorithm is finished). This last termination detection scheme is characterised in Theorem 6.12 that adds classical coverings-based symmetry breaking conditions to the characterisation of observed termination detection.

Such refinements of the notion of termination of a distributed algorithm are necessary to address all kind of termination that are encountered in distributed computing. One can think in particular about the composition of distributed algorithms where observed termination detection seems not enough decentralised.

For example, from Th. 6.11, it can be shown that they are no distributed algorithm with detection of the global termination for such computations – that are usually preliminary to general distributed tasks – like computing the degree of a node, or any computations that involve only a local part of the network (like in [NS95]). Indeed, on a huge network, without knowledge of something like a bound of the diameter, a node can not even know if a very distant node has ever started the distributed algorithm. Theorem 6.10 gives a characterisation when the task is uniform, i.e., when the same value has to be computed everywhere in the network. Open problems remains for this kind of termination detection.

Finally, we show that, as it seems intuitively, these notions form a strict hierarchy.

1.6 Summary

Section 2 reviews the definitions of coverings and quasi-coverings. It presents local computations and their relations with coverings and quasi-coverings. Section 3 presents local computations, coverings, quasi-coverings with their properties that we need in the sequel of the paper. Section 4 is devoted to the Mazurkiewicz algorithm, Szymanski, Shy and Prywes algorithm and some extensions. In Section 5, we define formally our four notions of termination detection (no detection, local termination, observed termination, global termination) and gives numerous examples. Our main results concerning the termination detection problem and the election problem are formulated and proved in Section 6 and Section 8. Section 7 presents some applications of the theorems that present classical network hypothesises as corollaries.

This paper is an extended and improved version of the extended abstracts [MT00] (the termination problem) and [GM02] (the election problem).

2 Basic Notions and Notations

2.1 Graphs

The notations used here are essentially standard [Ros00]. We only consider finite, undirected, connected graphs without multiple edges and self-loop. If G is a graph, then $V(G)$ denotes the set of vertices and $E(G)$ denotes the set of edges. Two vertices u and v are said to be adjacent if $\{u,v\}$ belongs to $E(G)$. The distance between two vertices u,v is denoted $d(u,v)$. The set of neighbours of v in G, denoted $N_G(v)$, is the set of all vertices of G adjacent to v. For a vertex v, we denote by $B_G(v)$ the ball of radius 1 with center v, that is the graph with vertices $N_G(v) \cup \{v\}$ and edges $\{\{u,v\} \in E(G) \mid u \in V(G)\}$. We also denote by $B_G(v,r)$ the ball of center v and radius $r \in \mathbb{N}$.

A homomorphism between G and H is a mapping $\varphi : V(G) \to V(H)$ such that if $\{u,v\}$ is an edge of G then $\{\varphi(u), \varphi(v)\}$ is an edge of H. Since we deal only with graphs without self-loop, we have $\varphi(u) \neq \varphi(v)$ whenever $\{u,v\}$ is an edge of G. Note also that $\varphi(N_G(u)) \subseteq N_H(\varphi(u))$. For an edge $\{u,v\}$ of G we define $\varphi(\{u,v\}) = \{\varphi(u), \varphi(v)\}$; this extends φ to a mapping $V(G) \cup E(G) \to V(H) \cup E(H)$. We say that φ is an isomorphism if φ is bijective and φ^{-1} is a homomorphism, too. We write $G \simeq G'$ whenever G and G' are isomorphic. A class of graphs will be any set of graphs containing all graphs isomorphic to some of its elements. The class of all graphs will be denoted $\mathcal{G}$.

For any set S, $\mathrm{card}(S)$ denotes the cardinality of S. For any integer q, we denote by $[1;q]$ the set $\{1,2,\ldots,q\}$.

2.2 Labelled Graphs

Throughout the paper we will consider graphs where vertices and edges are labelled with labels from a recursive alphabet L. A graph labelled over L will be denoted by (G,λ), where G is a graph and $\lambda : V(G) \cup E(G) \to L$ is the labelling

function. The graph G is called the underlying graph and the mapping λ is a labelling of G. For a labelled graph $(G;\lambda)$, $lab((G;\lambda))$ is the set of labels that occur in $(G;\lambda)$; i.e.,

$$lab((G;\lambda)) = \{\lambda(v) \mid v \in V(G)\}.$$

The class of labelled graphs over some fixed alphabet L will be denoted by $\mathcal{G}_L$. Note that since L is recursive, also $\mathcal{G}_L$ is recursive.

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

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

An occurrence of $(G;\lambda)$ in $(G';\lambda')$ is an isomorphism φ between $(G;\lambda)$ and a subgraph $(H;\lambda)$ of $(G';\lambda')$. It shall be denoted $\varphi : (G;\lambda) \to (G';\lambda')$.

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

2.3 Coverings

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

Examples and properties of coverings linked to networks are presented in [BL86,Bod89]. A generalization of coverings called fibrations has been studied by Boldi and Vigna in [BV02a], this paper emphasizes properties which found applications in distributed computing.

Example 2.1. Let R_n, $n > 2$, denote the ring on n vertices defined by $V(R_n) = [0;n-1]$ and $E(R_n) = \{\{x,y\} \mid y = x+1 \pmod n\}$. Let now $m \geq n$ and $\varphi_{m,n} : [0;m-1] \to [0;n-1]$ be the mapping defined by $\varphi_{m,n}(i) = i \pmod n$, for every $i \in [0;m-1]$. It is easy to check that for every $n > 2$ and for every $k > 2$, the ring $R_{k\,n}$ is a covering of the ring R_n via the mapping $\varphi_{k\,n,n}$.

The notion of covering extends to labelled graphs in an obvious way. The labelled graph $(H;\lambda')$ is covered by $(G;\lambda)$ via φ, if φ preserves labels and is a covering from G to H.

A graph G is called covering-minimal if every covering from G to some H is a bijection. Note that a graph covering is exactly a covering in the classical sense of algebraic topology, see [Mas91]. We have the following basic property of coverings [Rei32]:

Lemma 2.2. Let G be a covering of H via φ and let $v_1, v_2 \in V(G)$ be such that $v_1 \neq v_2$. If $\varphi(v_1) = \varphi(v_2)$ then $B_G(v_1) \cap B_G(v_2) = \emptyset$.

Lemma 2.3. Suppose that G is a covering of H via φ: Let T be a subgraph of H: If T is a tree then $\varphi^{-1}(T)$ is a set of disjoint trees, each isomorphic to T:

By considering simple paths between any two vertices, the previous lemma implies:

Lemma 2.4. For every covering φ from G to H there exists an integer q such that $\mathrm{card}(\varphi^{-1}(v)) = q$, for all $v \in V(H)$:

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

Example 2.5. A simple example of a 2-sheeted covering is given in Fig. 1. The image of each vertex of G is given by the corresponding Roman letter. Furthermore, we note that the image of each vertex is also given by its position on the H pattern (the spanning tree of H suggested in the figure). All examples of coverings below will be implicitely described by this geometric scheme, that is based on Theorem 2.6.

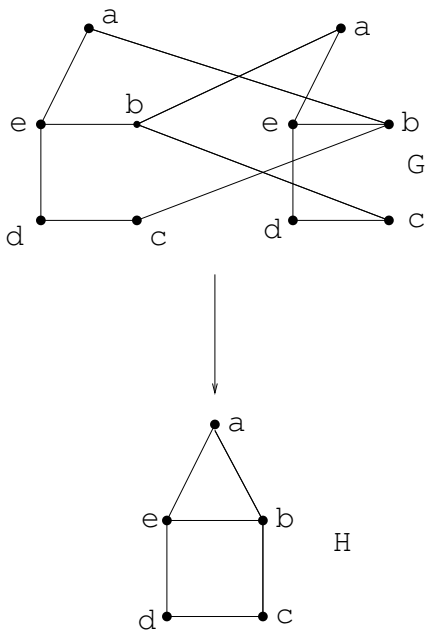


Fig. 1. The morphism φ is a covering from G to H.

Note also that for the rings $R_{k\ n}$ and R_n the number of sheets is k:

In [Rei32], it is shown that all coverings of H can be obtained from a given spanning tree of H:

Theorem 2.6 ([Rei32]). Let H be a graph and T a spanning tree of H. A connected graph G is a covering of H if and only if there exist a non-negative integer q and a set $\Phi = \{\varphi_{(x,y)} \mid x,y \in V(H); \{x,y\} \in E(H) \setminus E(T)\}$ of

permutations[1] on $[1;q]$ such that G is isomorphic to the graph $H_{T,}$ defined by:

$$V(H_{T,}) = \{(x,i) \mid x \in V(H), i \in [1,q]\},$$
$$E(H_{T,}) = \{\{(x,i),(y,i)\} \mid \{x,y\} \in E(T), i \in [1,q]\} \cup \{\{(x,i),(y, {}_{(x,y)}(i))\} \mid \{x,y\} \in E(H) \setminus E(T), i \in [1,q]\}.$$

The Universal Covering. The universal covering of a graph is a special example of covering. It may be defined as follows [Ang80,Lei82]. Let G be a graph, let v be a vertex of G; the universal covering of G, denoted $U(G)$, is the infinite tree whose vertex set is the set of all finite walks from v in G that do not traverse the same edge in two consecutive steps. Two nodes are adjacent if one is a one-step extension of the other. It is easy to verify that $U(G)$ is a tree, unique up to isomorphism and independent of the choice of v: Clearly $U(G)$ covers G: See Section 3.6 for a more formal definition.

2.4 Ambiguous Graphs and Coverings

In this part we give the definition of ambiguous graphs introduced by Mazurkiewicz in [Maz97] and we show that the non-ambiguous graphs are precisely the covering-minimal graphs.

A labelling is said to be locally bijective if vertices with the same label are not in the same ball and have isomorphic labelled neighbourhoods. Formally, we have:

Definition 2.7. [Maz97] Let L be a set of labels and let $(G,)$ be a labelled graph. The labelling $$ is locally bijective if it verifies the following two conditions:

1. For each $v \in V$ and for all $v', v'' \in B_G(v)$ we have $(v') = (v'')$ if and only if $v' = v''$.
2. For all $v', v'' \in V$ such that $(v') = (v'')$, the labelled balls $(B_G(v'),)$ and $(B_G(v''),)$ are isomorphic.

A graph G is ambiguous if there exists a non-bijective labelling of G which is locally bijective.

The labelling of the graph G in Figure 1 proves that G is ambiguous.

Locally bijective labellings and coverings are closely related through quotient graphs.

Definition 2.8. Let $$ be a labelling of the graph G. We define the quotient graph $G/$ by letting:

- $V(G/) = (V(G))$; and
- $E(G/) = \{\{, '\} \mid \exists v, v' \in V(G)$ such that $\{v,v'\} \in E(G), = (v), ' = (v')\}$.

[1] with the convention that ${}_{(x,y)} = {}^{-1}_{(y,x)}$

Lemma 2.9. Let G be a graph:

1. If α is a locally bijective labelling of G then the quotient mapping $G \to G/\alpha$ is a covering.
2. Every covering $\varphi : G \to H$ defines a locally bijective labelling of G:

Proof.

1. Using condition (1) in Definition 2.7 we note that G/α has no self-loop. Moreover, the conditions (1) and (2) imply that α is a bijection from $B_G(v)$ to $B_{G/\alpha}(\alpha(v))$, for each $v \in V(G)$. Hence $B_G(v)$ and $B_{G/\alpha}(\alpha(v))$ are isomorphic.
2. We consider $V(H)$ as set of labels and we label a vertex $v \in V(G)$ by $\varphi(v)$: It is straightforward to verify that this labelling is locally bijective.

Using the previous lemma we obtain:

Corollary 2.10. A graph is non-ambiguous if and only if it is covering-minimal.

3 Local Computations

In this section we recall the definition of local computations and their relation with coverings [LMZ95]. They model distributed algorithms on networks of processors of arbitrary topology. The network is represented as a connected, undirected graph where vertices denote processors and edges denote direct communication links. Labels (or states) are attached to vertices and edges.

Graph relabelling systems and more generally local computations satisfy the following constraints, that arise naturally when describing distributed computations with 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 applicability of a relabelling rule only depends on the local context of the relabelled subgraph.

The relabelling is performed until no more transformation is possible, i.e., until a normal form is obtained.

3.1 Local Computations

Local computations as considered here can be described in the following general framework. Let G_L be the class of L-labelled graphs and let $R \subseteq G_L \times G_L$ be a binary relation on G_L. Then R is called a graph rewriting relation. We assume

that R is closed under isomorphism, i.e., if $G \; R \; G^0$ and $H \simeq G$ then $H \; R \; H^0$ for some labelled graph $H^0 \simeq G^0$. In the remainder of the paper R^* stands for the reflexive-transitive closure of R: The labelled graph G is R-irreducible if there is no G^0 such that $G \; R \; G^0$: For $G \in \mathcal{G}_L$; $\mathrm{Irred}_R(G)$ denotes the set of R-irreducible (or just irreducible if R is fixed) graphs obtained from G using R^*; i.e., $\mathrm{Irred}_R(G) = \{H \mid G \; R^* \; H \text{ and } H \text{ is } R\text{-irreducible}\}$:

Definition 3.1. Let $R \subseteq \mathcal{G}_L \times \mathcal{G}_L$ be a graph rewriting relation.

1. R is a relabelling relation if whenever two labelled graphs are in relation then the underlying graphs are equal (we say equal, not just isomorphic), i.e.,
$$G \; R \; H \text{ implies that } G = H:$$
2. R is local if it can only modify balls of radius 1, i.e., $(G, \alpha) \; R \; (G, \alpha')$ implies that there exists a vertex $v \in V(G)$ such that
$$\alpha(x) = \alpha'(x) \text{ for every } x \notin V(B_G(v)) \cup E(B_G(v)):$$
The labelled ball $(B_G(v), \alpha)$ is a support of the relabelling relation.

The next definition states that a local relabelling relation R is locally generated if the applicability of any relabelling depends only on the balls of radius 1.

Definition 3.2. Let R be a relabelling relation. Then R is locally generated if it is local and the following is satisfied. For all labelled graphs (G, α), (G, α'), there exists vertice $v \in V(G)$, such that for all (H, β), (H, β'), $w \in V(H)$ such that the balls $B_G(v)$ and $B_H(w)$ are isomorphic via $\varphi : V(B_G(v)) \to V(B_H(w))$ and $\varphi(v) = w$, the following three conditions:

1. $\alpha(x) = \beta(\varphi(x))$ and $\alpha'(x) = \beta'(\varphi(x))$ for all $x \in V(B_G(v)) \cup E(B_G(v))$;
2. $\alpha(x) = \alpha'(x)$, for all $x \notin V(B_G(v)) \cup E(B_G(v))$;
3. $\beta(x) = \beta'(x)$, for all $x \notin V(B_H(w)) \cup E(B_H(w))$;

imply that $(G, \alpha) \; R \; (G, \alpha')$ if and only if $(H, \beta) \; R \; (H, \beta')$.

By definition, local computations on graphs are computations on graphs corresponding to locally generated relabelling relations.

We only consider recursive relabelling relations such that the set of irreducible graphs is recursive. The purpose of all assumptions about recursiveness done throughout the paper is to have "reasonable" objects w.r.t. the computational power. Furthermore, in order to prevent ambiguousness, Turing-computability will only be addressed as "recursivity", and we will restrict the use of the word "computability" to the context of local computations.

A sequence $(G_i)_{0 \leq i \leq n}$ is called an R-relabelling sequence (or relabelling sequence, when R is clear from the context) if $G_i \; R \; G_{i+1}$ for every $0 \leq i < n$ (with n being the length of the sequence). A relabelling sequence of length 1 is a relabelling step. The relation R is called noetherian on a graph G if there is no infinite relabelling sequence $G_0 \; R \; G_1 \; R \; \cdots$; with $G_0 = G$: The relation R is noetherian on a set of graphs if it is noetherian on each graph of the set. Finally, the relation R is called noetherian if it is noetherian on each graph.

3.2 Graph Relabelling Systems

We present now graph relabelling systems as used for modelling distributed algorithms, by describing the exact form of the relabelling steps. Each step will modify a star-graph, i.e., a graph with a distinguished center vertex connected to all other vertices (and having no other edge besides these edges). As any ball of radius one is isomorphic to a labelled star-graph, the support (or precondition) of any relabelling rule will be supposed to be a labelled star-graph.

Graph Relabelling Rules. A graph relabelling rule is a triple $r = (B_r, \lambda_r, \lambda'_r)$, where B_r is a star-graph and λ_r, λ'_r are two labellings of B_r. We refer to (B_r, λ_r) as the precondition of the rule r, whereas (B_r, λ'_r) is referred to as the relabelling through r.

Let $r = (B_r, \lambda_r, \lambda'_r)$ be a relabelling rule, H an (unlabelled) graph and λ, λ' two labellings of H. We say that (H, λ') is obtained from (H, λ) by applying the rule r to the occurrence φ of B_r in H (and we write $(H, \lambda) \underset{r,\varphi}{\Rightarrow} (H, \lambda')$) if the following conditions are satisfied, with v_0 denoting the center of B_r:

1. φ induces both an isomorphism from (B_r, λ_r) to $B_{(H,\lambda)}(\varphi(v_0))$ and from (B_r, λ'_r) to $B_{(H,\lambda')}(\varphi(v_0))$,
2. $\lambda'(x) = \lambda(x)$ for all $x \in (V(H) \setminus V(B_H(\varphi(v_0)))) \cup (E(H) \setminus E(B_H(\varphi(v_0))))$,

In this case we also say that φ is an occurrence of the rule r in (H, λ) and the image of B_r under φ is called the image of r under φ.

The relabelling relation $\underset{r}{\Rightarrow}$ induced by the rule r is defined by letting $(H, \lambda) \underset{r}{\Rightarrow} (H, \lambda')$ if there exists an occurrence φ of r in (H, λ) with $(H, \lambda) \underset{r,\varphi}{\Rightarrow} (H, \lambda')$.

Let $r = (B_r, \lambda_r, \lambda'_r)$ and $s = (B_s, \lambda_s, \lambda'_s)$ be two (not necessary distinct) relabelling rules and let

$$\varphi_r : (B_r, \lambda_r) \to (H, \lambda), \quad \varphi_s : (B_s, \lambda_s) \to (H, \lambda)$$

be two occurrences of r and s respectively in (H, λ). We say that these two occurrences overlap if

(i) the images of B_r by φ_r and B_s by φ_s have a common vertex, and
(ii) either $r \neq s$ or ($r = s$ and $\varphi_r \neq \varphi_s$).

Graph Relabelling Systems. A graph relabelling system is a recursive set R of graph relabelling rules, such that the set of labelled star-graphs that are preconditions of a rule in R is also recursive.

The relabelling relation $\underset{R}{\Rightarrow}$ is defined by $(G, \lambda) \underset{R}{\Rightarrow} (G, \lambda')$ if there is a rule $r \in R$ such that $(G, \lambda) \underset{r}{\Rightarrow} (G, \lambda')$.

Examples of graph relabelling systems are presented in [LMS99,LMZ95].

Clearly, graph relabelling systems represent locally generated relabelling relations. Conversely, any locally generated relabelling relation can be represented by a graph relabelling system.

Proposition 3.3. Let R be a relabelling relation. Then R is both locally generated and a recursive relation such that the set of irreducible graphs is recursive if and only if there exists a graph relabelling system R such that R equals $=_R$.

Proof. Given a locally generated relabelling relation R, we have to find a graph relabelling system R that generates R.

We define:

$$R = \{(B, \lambda, \lambda') \mid B \text{ is a star-graph}, (B, \lambda) R (B, \lambda')\}$$

First, R is obviously recursive since R is. The set of preconditions of R is also recursive, since one can check wether a precondition does not belong to the set of R-irreducible graphs. It is then straightforward to verify that R generates exactly R from Definition 3.2.

In the following, we do not discriminate between a locally generated relabelling relation and a graph relabelling system that generates it. They, both, model distributed algorithms.

Generic Rules. We explain here the convention under which we will describe graph relabelling systems later. If the number of rules is finite then we will describe all rules by their preconditions and relabellings. We will also describe a family of rules by a generic rule ("meta-rule"). In this case, we will consider a generic star-graph of generic center v_0 and of generic set of vertices $B(v_0)$. Within these conventions, we will refer to a vertex v of the star graph by writing $v \in B(v_0)$. If $\lambda(v)$ is the label of v in the precondition, then $\lambda'(v)$ will be its label in the relabelling. We will omit in the description labels that are not modified by the rule. This means that if $\lambda(v)$ is a label such that $\lambda'(v)$ is not explicitly described in the rule for a given v, then $\lambda'(v) = \lambda(v)$. In all the examples of graph relabelling systems that we consider in this paper the edge labels are never changed.

We do not require relabelling rules to be antisymmetric, but obviously a system with such rules would have some difficulties to terminate. Thus, in order to have light preconditions for generic rules, we consider that a rule (induced by a given generic rule) that would not modify any label in the star-graph is not enabled.

With these conventions, the only point we have to care about is to verify that the set of graph relabelling rules and the set of preconditions described by the generic rule are recursive.

Example Our first example is a $(d+1)$-coloring of regular graphs of degree d. This example will allow us to use the above described conventions.

Example 3.4. We consider the graph relabelling system $Colo_d$. The value of the label of a vertex v is denoted by $c(v)$. The "colors" used here are integers from $[1; d+1]$, all vertices are initially labeled by 0. The following generic rule means

that if v_0 is labelled by 0, then v_0 is relabelled by the smallest value that does not occur as label of one of its neighbours. The edge labels are not used in this example.

$Colo_d$: $(d+1)$-Coloring

Precondition :

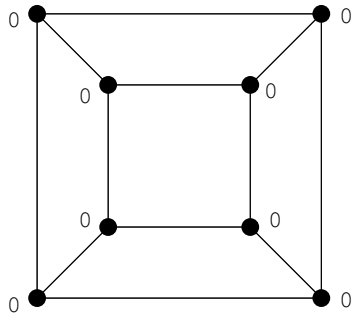
$c(v_0) = 0$

Relabelling :

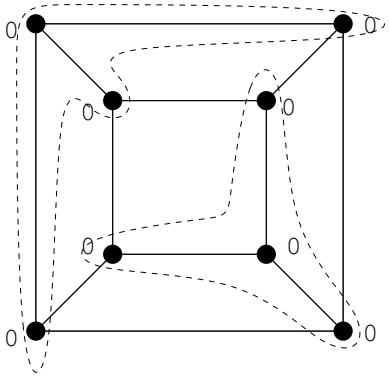
$c'(v_0) := \min([1;d+1] \setminus \{c(v) \mid v \in B(v_0); c(v) \neq 0\})$

The figures below show an execution of $Colo_3$.

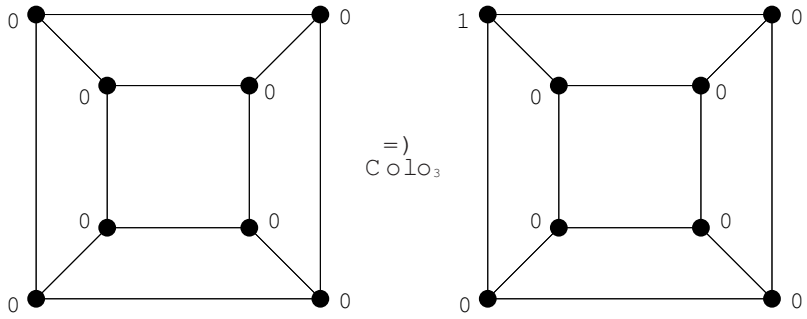
The initial labelling is the following:

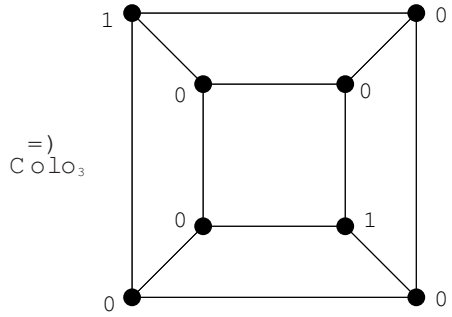


Two non-overlapping occurrences where a rule can be applied are indicated below :

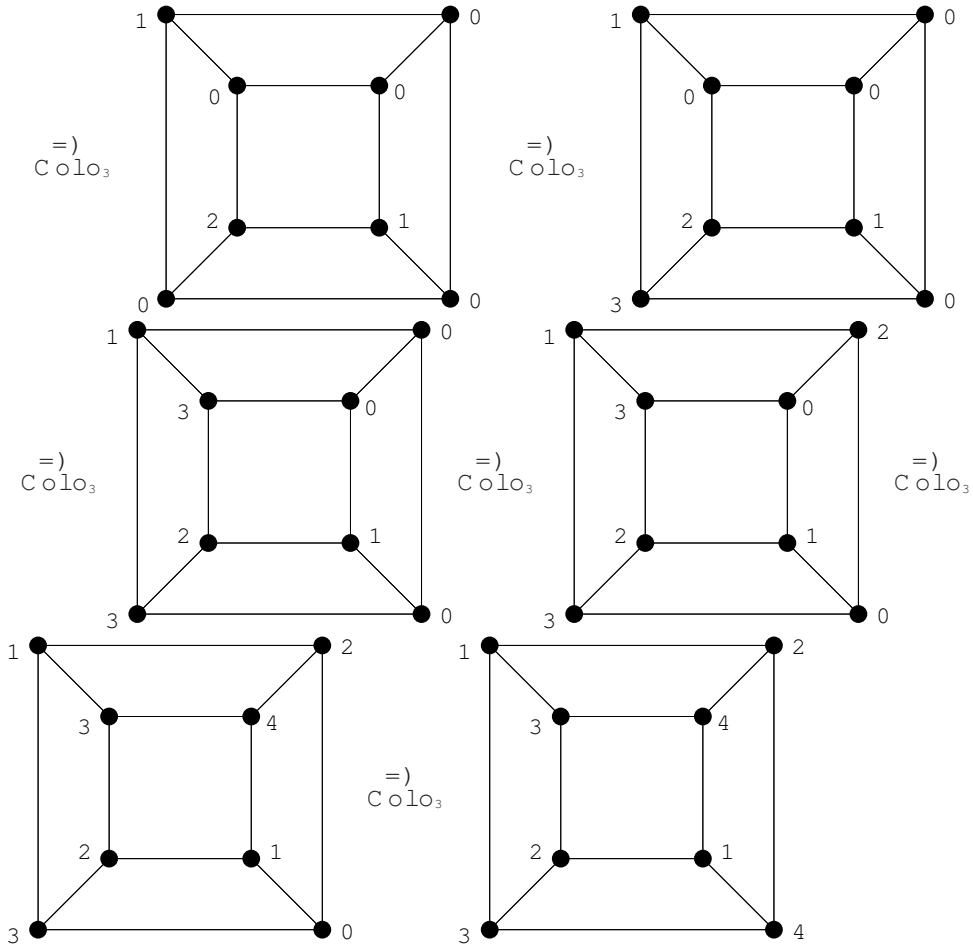


A corresponding relabelling sequence is as below :





The remaining part of the relabelling sequence is for instance:



One can note that the correctness of the algorithm follows from the fact that the set upon which the minimum is taken is never empty.

3.3 Distributed Computations of Local Computations

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

Hence, our notion of relabelling sequence associated to a locally generated relabelling relation may be regarded as a serialization [Maz87] of a distributed computation. This model is asynchronous, in the sense that several relabelling

steps may be done at the same time but we do not require that all of them have to be performed. In the sequel we will essentially handle sequential relabelling sequences, but the reader should keep in mind that such sequences may be done in parallel.

3.4 Local Computations and Coverings

We now present the fundamental lemma connecting coverings and locally generated relabelling relations. It states that whenever G is a covering of H, every relabelling step in H can be lifted to a relabelling sequence in G, which is compatible with the covering relation. It was first given in [Ang80].

Lemma 3.5 (Lifting Lemma). Let R be a locally generated relabelling relation and let G be a covering of H via φ: If $H \; R \; H'$ then there exists G' such that $G \; R \; G'$ and G' is a covering of H' via φ:

Proof. It suffices to show the claim for the case $H \; R \; H'$. Suppose that the relabelling step changes labels in $B_H(v)$; for some vertex $v \in V(H)$. We may apply this relabelling step to each of the disjoint labelled balls of $\varphi^{-1}(B_H(v))$, since they are isomorphic to $B_H(v)$. This yields G' which satisfies the claim.

This is depicted in the following commutative diagram:

$$\begin{array}{ccc} G & \xrightarrow{R} & G' \\ \downarrow{\scriptstyle \text{covering}} & & \downarrow{\scriptstyle \text{covering}} \\ H & \xrightarrow{R} & H' \end{array}$$

3.5 Local Computations and Quasi-coverings

We will see now a configuration where only relabelling chains of bounded length can be simulated. The notion of quasi-coverings was first introduced in [MMW 97] to prove impossibility of termination detection in some cases. However the definition of quasi-coverings here differs slightly from [MMW 97], providing new and simplified proofs, e.g., for Lemma 3.7 and Lemma 4.13. Here, the key parameter is the radius and not the size of the quasi-covering.

Definition 3.6. Let G, H be two labelled graphs and let φ be a partial function on $V(G)$ that assigns to each element of a subset of $V(G)$ exactly one element of $V(H)$: Then G is a quasi-covering of H via φ of radius r if there exists a finite or infinite covering G_0 of H via φ', vertices $z_0 \in V(G_0)$, $z \in V(G)$ such that:

1. $B_G(z,r)$ is isomorphic via ψ to $B_{G_0}(z_0,r)$,
2. the domain of definition of φ contains $B_G(z,r)$; and
3. $\varphi = \varphi'$ when restricted to $V(B_G(z,r))$.

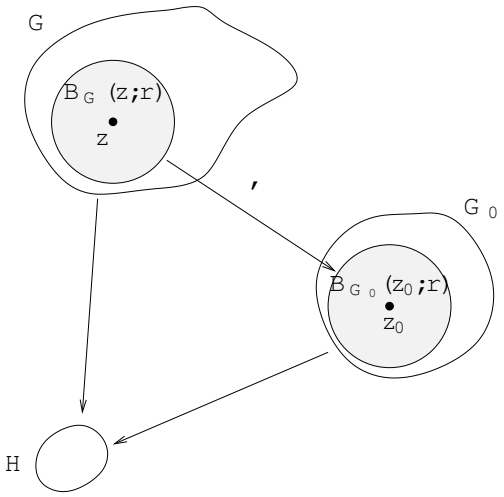


Fig.2. $\varphi : G \to H$ is a quasi-covering of radius r and associated covering $\varphi : G_0 \to H$.

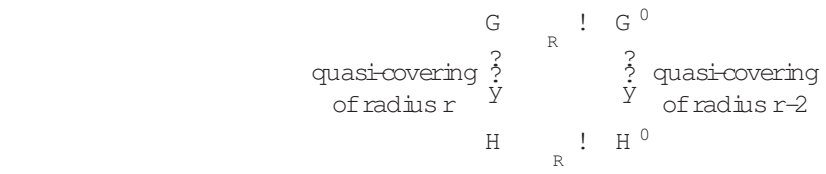
$\mathrm{card}(V(B_G(z;r)))$ is called the size of the quasi-covering, and z the center. The graph G_0 is called the associated covering of the quasi-covering. See Figure 2.

Quasi-coverings have been introduced to study the problem of the detection of the termination in [MMW 97]. The idea behind them is to enable the partial simulation of local computations on a given graph in a restricted area of a larger graph. The restricted area where we can perform the simulation will shrink while the number of simulated steps increases. The following lemma makes precise how much the radius shrinks when one step of simulation is performed:

Lemma 3.7 (Quasi-Lifting Lemma). Let R be a locally generated relabelling relation and let G be a quasi-covering of H of radius r via φ. Moreover, let $H \; R \; H'$. Then there exists G' such that $G \; R \; G'$ and G' is a quasi-covering of radius $r - 2$ of H':

Proof. Let G_0 be the associated covering and z be the center of the ball of radius r. Suppose now the relabelling step $H \; R \; H'$ applies rule R_0 and modifies labels in $B_H(v)$, for a given $v \in V(H)$. The rule R_0 can also be applied to all the balls $\varphi^{-1}(B_H(v))$ yielding G'_0 and φ'. It applied also to the balls $\varphi^{-1}(B_H(v))$ that are included in $B_G(z;r)$, since they are also isomorphic to $B_H(v)$. We get G' and φ' satisfying the quasi-covering properties with radius $r - 2$: consider w in $B_{G_0}(z; r-2)$: since any ball containing w is included in $B_G(z;r)$, w and $\varphi'(w)$ have the same label.

This is depicted in the following commutative diagram :



Using notation of this subsection:

Definition 3.8. We define the number of sheets q of a quasi-covering to be the minimal cardinality of the sets of preimages of vertices of H which are in the ball:

$$q = \min_{v \in V(H)} |\{w \in \alpha^{-1}(v) \mid B_K(w,1) \cap B_K(z_0,r)\}|.$$

With this definition, the notion of number of sheets is equivalent in the case of coverings.

Definition 3.9. A quasi-covering is strict if $B_G(z,r-1) \not\simeq G$:

Remark 3.10. A non strict quasi-covering is simply a covering.

Remark 3.11. With the same notation, if G is a strict quasi-covering of H of radius r then $|B_G(z,r)| \geq r$:

We have then the following technical lemma:

Lemma 3.12. Let G be a strict quasi-covering of H of radius r via α. For any $q \in \mathbb{N}$, if $r \geq q|V(H)|$ then α has at least q sheets.

Proof. Note K the associated covering. The quasi-covering being strict, we have that $|B_G(z,r)| \geq r \geq q|V(H)|$, hence $|V(K)| \geq q|V(H)|$. We deduce from Lemma 2.4 that K has at least q sheets.

Now, consider a spanning tree T of H rooted on $\alpha(z)$. Note T_1 the lifting of T rooted on z_0. By Theorem 2.6, there is $q-1$ distinct lifted spanning trees $T_2,\dots,T_q$ such that the subgraph induced by $T_1 \cup \dots \cup T_q$ is connected. As T has a diameter at most $|V(H)|-1$, we have that $T_1 \cup \dots \cup T_q \subset B_K(z_0, q|V(H)|)$. That means that every vertex of H has at least q preimages in $B_K(z_0,r)$, hence in $B_G(z,r)$.

The following expresses a link of the radius and of the size of the quasi-covering of a given graph.

Lemma 3.13. Let H be a graph with maximal degree d. Then for all quasi-covering of H of size s and radius r, we have

$$s \leq (d+1)^r.$$

Proof. Let G be a quasi-covering of H. Let z be the center, and r the radius. $B_G(z,r)$ is then a subgraph of maximal degree d. By induction, remarking that $|B(z,i+1) \setminus B(z,i)| \leq d|B(z,i)|$, we obtain that any ball of radius r and maximal degree d has a size at most $(d+1)^r$.

This bound is obviously not optimal but sufficient for our purpose. Remarking that a q-sheeted quasi-covering of a given graph H has a size greater than $q|V(H)|$, we get, from these two lemmas, a complete relation between the radius and the number of sheets of a quasi-covering.

3.6 Paths and Universal Coverings

A path is a sequence of neighbouring vertices in a graph.

Definition 3.14. A path from u_0 to u_n in a graph G is a sequence $\gamma = (u_0,\dots,u_n)$ such that for all i,

3.14.i $\{u_i, u_{i+1}\} \in E(G)$.

Furthermore, if, for all i,

3.14.ii $u_{i-1} \neq u_{i+1}$,

we say that γ is a non stuttering path [BV02a].

We denote by $\Gamma_G(u)$ the set of paths in G starting from vertex u. For any path $\gamma = (u_0,\dots,u_n)$ and any vertex v, we note $\gamma \cdot v$ the path $(u_0,\dots,u_n,v)$.

Definition 3.15. Let G be a (labelled) graph. Let u be a vertex of G. We denote by $\hat{\mathcal{G}}(u)$ the graph of non stuttering paths starting from u:

$$V(\hat{\mathcal{G}}(u)) = \{\gamma \in \Gamma_G(u) \mid \gamma \text{ is non stuttering}\};$$
$$E(\hat{\mathcal{G}}(u)) = \{\{\gamma, \gamma'\} \mid \gamma, \gamma' \in V(\hat{\mathcal{G}}(u)); \text{ and there exists a vertex } v \text{ of } G \text{ such that } \gamma' = \gamma \cdot v\}.$$

We denote by b the projection of $\hat{\mathcal{G}}(u)$ on G that maps any path to its final vertex.

Proposition 3.16. The graph $\hat{\mathcal{G}}(u)$ is a covering of G via the projection b.

Proof. Let v a vertex of G. Let a path $\gamma = (u_0,\dots,u_n)$ with $u_0 = u$ and $u_n = v$. Suppose that γ is not the empty path. By construction, γ has as neighbours $(u_0,\dots,u_{n-1})$. Being non stuttering, it also has as neighbours the paths of the set $\{\gamma \cdot w \mid w \in N(v), w \neq u_{n-1}\}$. Hence b defines an isomorphism from $B_{\hat{G}(u)}(\gamma)$ to $B_G(v)$. If γ is the empty path, the proof is obvious.

For all vertices u, v, $\hat{\mathcal{G}}(u)$ is isomorphic to $\hat{\mathcal{G}}(v)$ [BV02a]. We shall denote by $\hat{\mathcal{G}}$ this graph defined up to isomorphism. We say that $\hat{\mathcal{G}}$ is the universal covering of G.

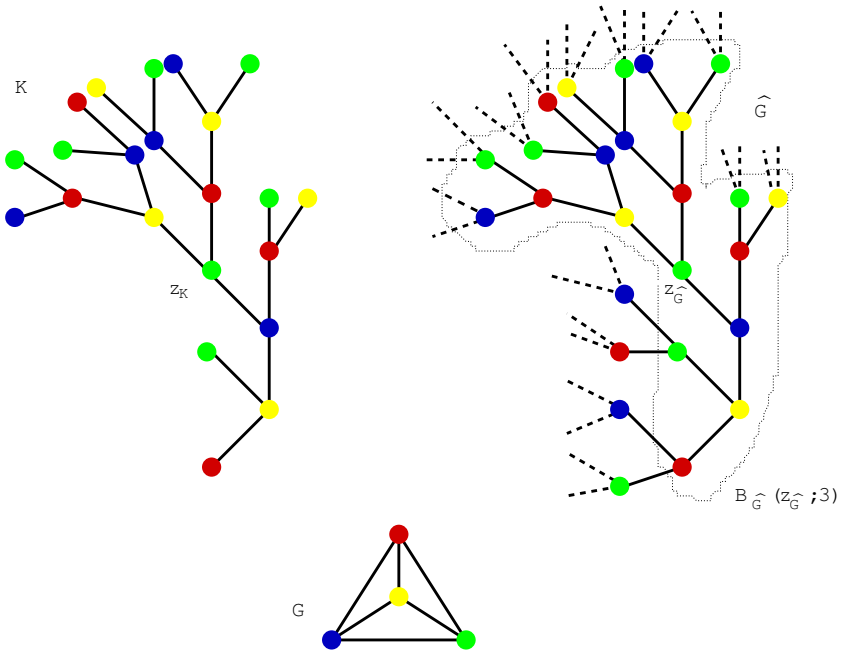


Fig. 3. K is a quasi-covering of radius 3 of G, obtained by truncation of $\widehat{G}$

Remark 3.17. This (possibly infinite) tree provides numerous examples of quasi-coverings. For a given graph G, by truncation of the universal covering $\tilde{G}$ to a ball of given radius, we obtain quasi-coverings of G. See Fig. 3.

The Reidemeister Theorem (Th. 2.6) is another tool to easily build quasi-covering of arbitrary radius.

3.7 Extension of Locally Generated Relabelling Relations

In this subsection, we show how the properties of a graph relabelling relation on a family $\mathcal{F}$ can be naturally extended to the family of graphs that are covered by a graph of $\mathcal{F}$.

Definition 3.18. Let $\mathcal{F}$ be a graph family. We note $\tilde{\mathcal{F}}$ the family of graphs that are covered by a graph of $\mathcal{F}$.

$$\tilde{\mathcal{F}} = \{H \mid \exists G \in \mathcal{F}, G \text{ is a covering of } H\}.$$

Note that $\mathcal{F}$ is a subset of $\tilde{\mathcal{F}}$. The first easy property is that if a R is noetherian on $\mathcal{F}$, it is also noetherian on $\tilde{\mathcal{F}}$.

Lemma 3.19. Let R be a relabelling system. If R is notetherian on $\mathcal{F}$, it is also noetherian on $\tilde{\mathcal{F}}$.

Proof. Suppose there is an infinite relabelling chain on $H \in \tilde{\mathcal{F}}$. Note G a graph in $\mathcal{F}$ that is a covering of H. By the Lifting Lemma, we get an infinite relabelling chain on G. Hence a contradiction.

Remark 3.20. The closure under covering of a recursive graph family is not necessarily recursive. Consider the following family

$$\mathcal{F}_c = \{G \mid G \text{ is a ring and there exists } p, i, m \in \mathbb{N} \text{ such that}$$
$$p^m \text{ is the size of } G;$$
$$p \text{ is the } i\text{-th prime number};$$
$$\text{Turing Machine number } i \text{ has halted before step } m\}.$$

The family $\mathcal{F}_c$ is obviously recursive and $\tilde{\mathcal{F}}_c$ is obviously non recursive: it is straighforward to see that deciding if a ring of prime size can be lifted in $\mathcal{F}_c$ corresponds to the Halting Problem for Turing Machines.

4 Fundamental Algorithms

In this section, we present our two fundamental algorithms.

4.1 Mazurkiewicz' Enumeration Algorithm

A distributed enumeration algorithm on a graph G is a distributed algorithm such that the result of any computation is a labelling of the vertices that is a bijection from $V(G)$ to $\{1,2,\ldots,|V(G)|\}$. In particular, an enumeration of the vertices where vertices know whether the algorithm has terminated solves the election problem. In [Maz97] Mazurkiewicz presents a distributed enumeration algorithm for covering-minimal (non-ambiguous) graphs.

The computation model in [Maz97] consists exactly in relabelling balls of radius 1 and the initial graph is unlabelled.

Mazurkiewicz' algorithm will be denoted M. By abuse of language we still speak of an enumeration algorithm, even when it is applied to ambiguous graphs (for which no enumeration algorithm exists, [Maz97]). The final labellings that are incorrect from the enumeration point of view have interesting properties in the context of local computation. Namely, they determine a graph that is covered by the input graph.

In the following we describe Mazurkiewicz' algorithm including its extension to labelled graphs.

Enumeration Algorithm. We first give a general description of the algorithm M applied to a graph G: Let $G = (G, \ell)$ and consider a vertex v_0 of G, and the set $\{v_1,\ldots,v_d\}$ of neighbours of v_0:

The label of the vertex v_0 used by M is the pair $(\ell(v_0), c(v_0))$ where $c(v_0)$ is a triple $(n(v_0), N(v_0), M(v_0))$ representing the following information obtained during the computation (formal definitions are given below):

- $n(v_0) \in \mathbb{N}$ is the number of the vertex v_0 computed by the algorithm,
- $N(v_0) \in \mathbb{N}$ is the local view of v_0, and it is either empty or a family of triples defined by:
$$\{(n(v_i), \ell(v_i), (\{v_0, v_i\}, \ell))_{1 \le i \le d}\},$$
- $M(v_0) \in L \times \mathbb{N} \times \mathbb{N}$ is the mailbox of v_0 and contains the whole information received by v_0 at any step of the computation.

Each vertex v attempts to get its own number $n(v)$, which will be an integer between 1 and $|V(G)|$. A vertex chooses a number and broadcasts it together with its label and its labelled neighbourhood all over the network. If a vertex u discovers the existence of another vertex v with the same number, then it compares its label and its local view, i.e., its number-labelled ball, with the local view of its rival v. If the label of v or the local view of v is "stronger", then u chooses another number. Each new number, with its local view, is broadcast again over the network. At the end of the computation it is not guaranteed that every vertex has a unique number, unless the graph is covering-minimal. However, all vertices with the same number will have the same label and the same local view.

The crucial property of the algorithm is based on a total order on local views such that the local view of any vertex cannot decrease during the computation.

We assume for the rest of this paper that the set of labels L is totally ordered by $<_L$. Consider a vertex v_0 with neighbourhood $\{v_1, \ldots, v_d\}$ and assume that:

- $n(v_1) \leq n(v_2) \leq \cdots \leq n(v_d)$;
- if $n(v_i) = n(v_{i+1})$ then $\alpha(v_i) \leq_L \alpha(v_{i+1})$;
- if $n(v_i) = n(v_{i+1})$ and $\alpha(v_i) = \alpha(v_{i+1})$ then $\beta(\{v_0, v_i\}) \leq_L \beta(\{v_0, v_{i+1}\})$.

Then the local view $N(v)$ is the d-tuple

$$((n(v_1), \alpha(v_1), \beta(\{v_0, v_1\})), \ldots, (n(v_d), \alpha(v_d), \beta(\{v_0, v_d\}))).$$

Let $N_>$ be the set of all such ordered tuples. We define a total order $\prec$ on $N_>$ by comparing the numbers, then the vertex labels and finally the edge labels. Formally, for two elements

$$((n_1, l_1, e_1), \ldots, (n_d, l_d, e_d)) \text{ and } ((n'_1, l'_1, e'_1), \ldots, (n'_{d'}, l'_{d'}, e'_{d'}))$$

of $N_>$ we define

$$((n'_1, l'_1, e'_1), \ldots, (n'_{d'}, l'_{d'}, e'_{d'})) \prec ((n_1, l_1, e_1), \ldots, (n_d, l_d, e_d))$$

if one of the following conditions holds:

1. $n_1 = n'_1, \ldots, n_{i-1} = n'_{i-1}$ and $n'_i < n_i$ for some i,
2. $d' < d$ and $n_1 = n'_1, \ldots, n_{d'} = n'_{d'}$,
3. $d = d'$, $n_1 = n'_1, \ldots, n_d = n'_d$ and $l_1 = l'_1, \ldots, l_{i-1} = l'_{i-1}$ and $l'_i <_L l_i$ for some i,
4. $d = d'$ and $n_1 = n'_1, \ldots, n_d = n'_d$ and $l_1 = l'_1, \ldots, l_d = l'_d$ and $e_1 = e'_1, \ldots, e_{i-1} = e'_{i-1}$ and $e'_i <_L e_i$ for some i.

If $N(u) \prec N(v)$, then we say that the local view $N(v)$ of v is stronger than the one of u. The order $\prec$ is a total order on $N = N_> \cup \{\alpha, \beta\}$ with, by definition, $\alpha \prec \beta \prec N$ for every $N \in N_>$.

We now describe the algorithm through a graph relabelling system. The initial labelling of the vertex v_0 is $(\alpha(v_0), (0, \emptyset, \emptyset, \emptyset))$.

The rules are described below for a given ball $B(v_0)$ with center v_0. The vertices v of $B(v_0)$ have labels $(\alpha(v), (n(v), N(v), M(v)))$. The labels obtained after applying a rule are $(\alpha(v), (n'(v), N'(v), M'(v)))$. We recall that we omit labels that are unchanged.

M{1: Diffusion rule

Precondition:

There exists $v \in B(v_0)$ such that $M(v) \not\subseteq M(v_0)$.

Relabelling:

For all $v \in B(v_0)$, $M'(v) := \bigcup_{w \in B(v_0)} M(w)$.

M {2 : Renaming rule

Precondition :

For all $v \in B(v_0), M(v) = M(v_0)$.

$(n(v_0) = 0)$ or

$(n(v_0) > 0$ and there exists $(l, n(v_0), N) \in M(v_0)$ such that

$(\alpha(v_0) < l)$ or $((\alpha(v_0) = l)$ and $(N(v_0) \neq N)))$.

Relabelling :

$n'(v_0) = 1 + \max\{n \in \mathbb{N} \mid (l, n, N) \in M(v_0)$ for some $l, N\}$.

For every $v \in B(v_0)$, $N'(v)$ is obtained from $N(v)$ by replacing the value of $n(v_0)$ by $n'(v_0)$:

For every $v \in B(v_0)$, the mailbox contents $M(v)$ changes to

$M'(v) = M(v) \cup \{(\alpha(w), n'(w), N'(w)) \mid w \in B(v_0)\}$.

4.2 Properties of Mazurkiewicz' Algorithm

In order to make the paper self-contained, we present a complete proof of the correctness of Mazurkiewicz' algorithm in our framework following the ideas developed in [Maz97].

Let G be a labelled graph. If v is a vertex of G then the label of v after a run of Mazurkiewicz' algorithm is denoted $(\alpha(v), c(v))$ with $c(v) = (n(v), N(v), M(v))$ and (α, c) denotes the final labelling.

Theorem 4.1. [Maz97] *Any run of Mazurkiewicz' enumeration algorithm on a connected labelled graph $G = (G, \lambda)$ terminates and yields a final labelling (α, c) verifying the following conditions for all vertices $v, v' \in G$:*

4.1.i *Let m be the maximal number in the final labelling, $m = \max_{v \in V(G)} n(v)$. Then for every $1 \leq p \leq m$ there is some $v \in V(G)$ with $n(v) = p$.*

4.1.ii $M(v) = M(v')$.

4.1.iii $(\alpha(v), n(v), N(v)) \in M(v')$.

4.1.iv *Let $(l, n, N) \in M(v')$. Then $\alpha(v) = l$, $n(v) = n$ and $N(v) = N$ for some vertex v if and only if there is no pair $(l', n, N') \in M(v')$ with $l <_L l'$ or $(l = l'$ and $N \neq N')$.*

4.1.v $n(v) = n(v')$ *implies* $(\alpha(v) = \alpha(v')$ and $N(v) = N(v'))$

4.1.vi n *induces a locally bijective labelling of G.*

We first prove the following lemmas. We say that a number m is known by v if $(l, m, N) \in M(v)$ for some l and some N. In the following i is an integer denoting a computation step. Let $(\alpha(v), (n_i(v), N_i(v), M_i(v)))$ be the label of the vertex v after the ith step of the computation.

Lemma 4.2. *For each v, i:*

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

Proof. The property is obviously true for the vertices that are not involved in the rule applied at step i: For the other vertices we note that the renaming rule applied to v_0 increments $n_i(v_0)$, adds elements to some mailboxes and makes some $N(u)$ stronger. Moreover the diffusion rule only adds elements to mailboxes.

The fact that $N_i(v) \subseteq N_{i+1}(v)$ comes from the definition of $\preceq$. In other words, this order ensures that the past local views of a vertex are always weaker than its present one.

Furthermore, one of the inequalities is strict for at least one vertex, namely the one for which the previous rule was applied.

Lemma 4.3. For every $v \in V(G)$ and $(l,m,N) \in M_i(v)$ there exists a vertex $w \in V(G)$ such that $n_i(w) = m$.

Proof. Assume that the number m is known by v and let $U = \{u \in V(G) \mid \exists j < i, n_j(u) = m\}$. Obviously U is not empty. Let $w \in U$ and let $j < i$ such that

1. $n_j(w) = m$;
2. for any $u \in U$ and for any $k < i$ verifying $n_k(u) = m$ we have: $N_k(u) \subseteq N_j(w)$.

Clearly, the renaming rule cannot be applied to w, hence $n_i(w) = m$.

Next, we claim that whenever a number is known, all positive smaller numbers are assigned to some vertex.

Lemma 4.4. For every vertex $v \in V(G)$ such that $n_i(v) \neq 0$ and for every $m \in [1; n_i(v)]$, there exists some vertex $w \in V(G)$ such that $n_i(w) = m$.

Proof. We show this claim by induction on i. At the initial step ($i = 0$) the assertion is true. Suppose that it holds for $i \geq 0$. If the diffusion rule is used, the assertion is true for $i+1$. If the renaming rule is applied to v_0 then we just have to verify it for v_0, and more precisely for all numbers m in the interval $\{n_i(v_0); n_i(v_0)+1; \ldots; n_{i+1}(v_0)\}$. The property holds obviously for $n_{i+1}(v_0)$ and, being known by v_0 at step $i+1$, the property for $n_i(v_0)$ is a consequence of Lemma 4.3.

If the interval $\{n_i(v_0)+1; \ldots; n_{i+1}(v_0)-1\}$ is empty then the condition is obviously satisfied. Otherwise by definition of the renaming rule, $n_{i+1}(v_0) - 1$ is known by v_0 at step i and thus Lemma 4.3 implies that there exists $w \neq v_0$ such that $n_i(w) = n_{i+1}(v_0) - 1$. For every $m \in \{n_i(v_0)+1; \ldots; n_{i+1}(v_0)-1\}$, we have, by induction hypothesis on w that there exists a vertex $x \in V(G)$ such that $n_i(x) = m$. For every such x, because v_0 is the only vertex changing its name from step i to $i+1$, $n_i(x) = n_{i+1}(x)$, which proves the assertion for step $i+1$.

We show now Theorem 4.1:

Proof.

As before, we denote by $(\ (v); (n_i(v); N_i(v); M_i(v)))$ the label of the vertex v after the ith step of the computation.

As there are no more than $|V(G)|$ different numbers assigned it follows from Lemma 4.2 and from Lemma 4.4 that the algorithm terminates.

The properties 1 to 6 of the final labelling are easily derived from the above part of the proof.

1. By Lemma 4.4 applied to the final labelling.
2. Otherwise, the diffusion rule could be applied.
3. A direct corollary of the previous property.
4. We have obtained a final labelling, thus it is a direct consequence of the diffusion rule and of the precondition of the renaming rule.
5. A direct consequence of the previous point.
6. The first part of Definition 2.7 is a consequence of the rewriting mechanism: when a vertex v is numbered, its number is put in mailboxes of adjacent vertices. Thus vertices at distance 2 of v cannot have the same number as v: The second part of Definition 2.7 is a consequence of the precondition of the renaming rule: the renaming rule could have been applied to vertices having the same number and non-isomorphic local views.

This ends the proof of the theorem.

Remark 4.5. By points 1 and 6 of Theorem 4.1, and similarly to [Maz97], the algorithm computes for non-ambiguous graphs (and thus for minimal graphs by Corollary 2.10), a one-to-one correspondence n between the set of vertices of G and the set of integers $\{1;\dots;|V(G)|\}$.

4.3 Toward an Enhanced Mazurkiewicz' Algorithm

In this section we prove that even by applying Mazurkiewicz' algorithm to a graph G that is not covering-minimal, we can get some relevant information. In this case, we prove that we can interpret the mailbox of the final labelling as a graph H that each vertex can compute and such that G is a covering of H:

For a mailbox M, we define the graph of the "strongest" vertices as follows. First, for $l \in L; n \in N; N \in \mathbb{N}; M \subseteq L \times N \times \mathbb{N}$, we define the predicate $\underline{\text{Strong}}(l;n;N;M)$ that is true if there is no $(l';n;N') \in M$ verifying

$$l' > l \text{ or } (l = l' \text{ and } N < N').$$

The graph H_M of strongest vertices of M is then defined by

$$\begin{aligned} V(H_M) &= \{n \mid \exists N; l : \underline{\text{Strong}}(l;n;N;M)\}; \\ E(H_M) &= \{\{n;n'\} \mid \exists N; l : \underline{\text{Strong}}(l;n;N;M); \text{ and } \exists l';l'' : \\ &\qquad N = (\dots;(n';l';l'');\dots)\}. \end{aligned}$$

We also define a labelling on this graph by $\lambda_M(n) = (n;l;N;M)$; with $\underline{\text{Strong}}(n;l;N;M)$ for some N; and $\lambda_M(\{n;n'\}) = l''$; with $\underline{\text{Strong}}(n;l;N;M)$ and $N = (\dots;(n';l';l'');\dots)$.

The uniqueness of this definition comes from the definition of $\underline{\text{Strong}}$ and from Theorem 4.1.v.

Let σ be a run of M: Then $(H_{M(u)}; \sigma_{M(u)})$ does not depend on u by Theorem 4.1.2. We then define $\tau(G) = (H_{M(u)}; \sigma_{M(u)})$; for any vertex u. Finally, we have:

Proposition 4.6. For a given execution σ of Mazurkiewicz algorithm, we have

$$V(\tau(G)) = \{\eta(v) | v \in V(G)\};$$

$$E(\tau(G)) = \{\{\eta(v), \eta(w)\} | \{v, w\} \in E(G)\}:$$

Remark 4.7. Before we emphasize the role of $\tau(G)$, note that $\tau(G)$ can be locally computed by every vertex, and that the graph depends only on the label M.

The next proposition states that we can see a run of M as computing a graph covered by G: Conversely, as a "translation" from [Maz97, Th. 5], every graph covered by G can be obtained by a run of the algorithm.

Proposition 4.8. Let G be a labelled graph.

1. For all runs σ of M, G is a covering of $\tau(G)$:
2. (completeness) For all H such that G is a covering of H, there exists a run σ such that $H \simeq \tau(G)$:

Proof.

1. Since η is locally bijective (Theorem 4.1.6), we obtain from Lemma 2.9 that G is a covering of $\tau(G)$:
2. We exhibit a run of M having the required property. Suppose that we have an enumeration of the vertices of H. Let λ be the labelling of G obtained by lifting the enumeration. There is an execution of Mazurkiewicz' algorithm such that each vertex v of G gets $\lambda(v)$ as a final η-labelling.
   This is done in the following way. First we apply the renaming rule to all vertices in $\lambda^{-1}(1)$. This is possible because there is no overlapping of balls, since G is a covering of H. Then we apply the diffusion rule as long as we can. After that, we apply the renaming rule to $\lambda^{-1}(2)$. Because of the diffusion, the number 1 is known by all the vertices, so the vertices of $\lambda^{-1}(2)$ get labelled by 2. And so on, until each vertex v gets labelled by $\lambda(v)$.

From Proposition 4.8.1, we can see a run of M as computing a covering. Furthermore, if the underlying graph is covering-minimal, then $\tau(G)$ is an isomorphic copy of G. This copy can be computed from their mailbox by any vertex, providing a "map" { with numbers of identification { of the underlying network. Thus, on minimal networks, the algorithm of Mazurkiewicz can actually be seen as a cartography algorithm.

Interpretation of the Mailboxes at the Step i: The previous results concern the interpretation of the final mailboxes. Now, we consider a relabelling chain $(G_i)_{0 \leq i \leq}$: For a given i and a given vertex v we prove that it is possible to interpret the label of v in G_i as a graph quasi-covered by G_i: We recall notation. Let G be a labelled graph. Let $\mathcal{A}$ be a run of the Mazurkiewicz algorithm and let $(G_i)_{0 \leq i \leq}$ be a chain associated to $\mathcal{A}$ with $(G_0 = G)$: If v is a vertex of G then the label of v at step i is denoted by $\mathcal{A}(v); c_i(v)) = (\mathcal{A}(v); (n_i(v); N_i(v); M_i(v)))$: Using the interpretation of the previous section by defining $Strong(M_i(v))$; this label enables in some cases the reconstruction of the graph $H_{M_i(v)}$: We note

$$H_i(v) = \begin{cases} H_{M_i(v)} \text{ if it is defined and } (n_i(v); \mathcal{A}(v); N_i(v)) \in Strong(M_i(v)) \\ ? \text{ otherwise.} \end{cases}$$

We prove that G_i is a quasi-covering of $H_i(v)$: First, we need a definition:

Definition 4.9. Let $(G_i)_{0 \leq i \leq}$; be a relabelling chain obtained with the Mazurkiewicz algorithm and let v be a vertex. We associate to the vertex v and to the step i the integer $r^{(i)}_{agree}(v)$ being the maximal integer bounded by the diameter of G such that any vertex w of $B(v; r^{(i)}_{agree}(v))$ verifies: $H_i(v) = H_i(w)$:

Now we can state the main result of this section:

Theorem 4.10. Let $(G_i)_{0 \leq i \leq}$; be a relabelling chain obtained with the Mazurkiewicz algorithm and let v be a vertex. The graph G_i is a quasi-covering of $H_i(v)$ centered on v of radius $r^{(i)}_{agree}(v)$:

Proof. Let $r = r^{(i)}_{agree}(v)$; and let φ be the partial function which associates to the vertex u of $B_{G_i}(v; r)$ the vertex $n_i(u)$: The aim of the proof is to verify that G_i is a quasi-covering via φ of $H_i(v)$ centered on v of radius r:

Using notation of the definition of quasi-coverings, first we define the covering K: Let B be an isomorphic copy of $B_{G_i}(v; r)$: The graph K is obtained by adding to B infinite trees defined as follows.

Let U be the universal covering of $H_i(v)$: Let x be a vertex of $H_i(v)$; let S verifying $S \subseteq N_{G_i(v)}(x)$; we define $U(x; S)$ as the subtree of U obtained by considering walks rooted in x such that the first step is of the form $\{x; s\}$ with $s \in S$:

For each vertex w such that $d(v; w) = r$; we define U_w as an isomorphic copy to $U(\mathcal{A}(w); S_w)$ with

$$S_w = \{s \in N_{G_i(v)}(\mathcal{A}(w)) \mid \forall y \in N_{G_i}(w) \cap B_{G_i}(v; r) \quad \varphi(y) \neq s\}:$$

The copies are disjoint, i.e., if $w \neq w'$ then $V(U_w) \cap V(U_{w'}) = \emptyset$:

For each vertex w such that $d(v; w) = r$; we add U_w to B by identifying the copy of w and the root of U_w: Let K be the graph we have built by this way.

The isomorphism φ' is the canonical bijection between $B_{G_i}(v; r)$ and B:

We define the morphism φ from K to $H_i(v)$ by:

- if $u \in B$ then $\varphi(u) = \varphi(\varphi'^{-1}(u))$; and

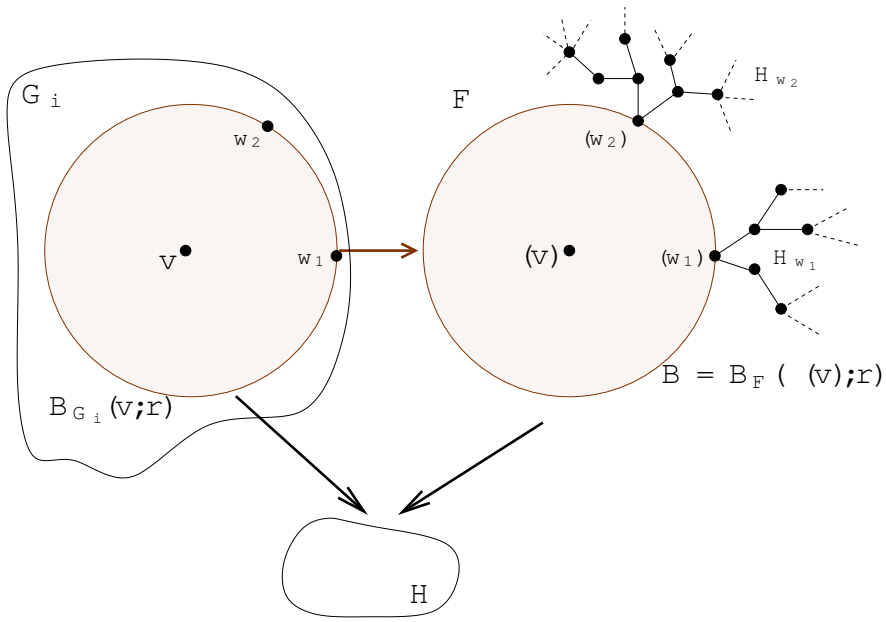


Fig. 4. Construction of the associated covering F.

- if $u \in U_w$ and t is the end-vertex of the path in $H_i(v)$ corresponding to u then $(u) = t$:

First, we verify that $$ is a morphism. There are three cases:

- if u and v are adjacent in U_w then by construction (u) and (v) are adjacent,
- if u and v are adjacent in B then they are adjacent in $H_{M_i(u)} = H_{M_i(v)}$ thus (u) and (v) are adjacent in $H_i(v)$;
- if u and v are adjacent and u belongs to B and v belongs to U_w for some w; by construction of U_w the vertices (u) and (v) are adjacent in $H_i(v)$:

By construction, $$ is surjective. To achieve the proof we verify that for all vertices u the restriction of $$ to $N_K(u)$ is a bijection onto $N_{H_i(v)}$:
Once more there are three cases.

- If $u \in H_w$ for some vertex w and if u is not the root of H_w then, by definition of the universal covering, the restriction of $$ to $N_K(u)$ is a bijection onto $N_{H_i(v)}$:
- If $u \in B_B(\varphi(v); r-1)$: We prove that the restriction of $$ to $N_{G_i}(\varphi^{-1}(u))$ is a bijection onto $N_{G_i(v)}((u))$: By definition of $$ the restriction is surjective; furthermore two vertices in the same ball of radius 1 have different numbers (it is a direct consequence of the Mazurkiewicz algorithm) thus the restriction is also injective.

- If $d(u,\varphi(v)) = r$ then using the same argument that for the previous item combined with the definition of the universal covering we obtain the result.

Remark 4.11. The previous result remains true for any radius bounded by $r^{(i)}_{agree}(v)$.

4.4 An Algorithm to Detect Stable Properties

In this section we describe in our framework the algorithm by Szymanski, Shy and Prywes (the SSP algorithm for short) [SSP85].

The SSP Algorithm We consider a distributed algorithm which terminates when all processes reach their local termination conditions. Each process is able to determine only its own termination condition. The SSP algorithm detects an instant in which the entire computation is achieved.

Let G be a graph, to each node v is associated a predicate $P(v)$ and an integer $a(v)$. Initially $P(v)$ is false and $a(v)$ is equal to -1.

The relabelling rules are the following, let v_0 be a vertex and let $\{v_1,\ldots,v_d\}$ the set of vertices adjacent to v_0. If $P(v_0) = false$ then $a(v_0) = -1$; if $P(v_0) = true$ then $a(v_0) = 1 + Min\{a(v_k) \mid 0 \leq k \leq d\}$.

A Generalization of the SSP Algorithm We present here a generalization of the hypothesis under which the SSP rules are run. For every vertex v, the value of $P(v)$ is no more a boolean and can have any value. Hence we will talk of the valuation P. Moreover, we do not require each process to determine when it reachs its own termination condition. Moreover the valuation P must verify the following property: for any α, if $P(v)$ has the value α and changes to $\beta \neq \alpha$ then it cannot be equal to α at an other time. In other words, under this hypothesis, the function is constant between two moments where it has the same value. We say that the valuation P is value-convex.

We extend the SSP rules and we shall denote by G_SSP this generalisation. In G_SSP, the counter of v is incremented only if P is constant on the ball $B(v)$. As previously, every underlying rule that computes in particular $P(v)$; has to be modified in order to eventually reinitialize the counter. Initially $a(v) = -1$ for all vertices. The G_SSP rule modifies the counter a.

Rule_for_G_SSP : Modified rule for G_SSP
  Precondition :
    ...
    unchanged
    ...
  Relabelling :
    ...
    unchanged

...
For every vertex v of $B(v_0)$,
if $P'(v) \neq P(v)$ then
$a'(v) := -1$;
otherwise
$a'(v) := a(v)$;

GSSP : GSSP rule
Precondition :
For all $v \in B(v_0)$, $P(v) = P(v_0)$,
Relabelling :
$a'(v_0) := 1 + \min\{a(v) \mid v \in B(v_0)\}$.

We shall now use the following notation. Let $(G_i)_{0 \leq i}$ be a relabelling chain associated to the algorithm GSSP. We denote by $a_i(v)$ (resp. $P_i(v)$) the value of the counter (resp. of the function) associated to the vertex v of G_i. According to the definition of the GSSP rule, we remark that for every vertex v, $a(v)$ can be increased, at each step, by 1 at most and that if $a(v)$ increases from h to $h+1$, that means that at the previous step, all the neighbours w of v were such that $a(w) \geq h$ and $P(w) = P(v)$. The following lemma is the iterated version of this remark.

Lemma 4.12. For all j, for all v, for all $w \in B(v, a_j(v))$, there exists an integer $i \leq j$ such that

$$a_i(w) \geq a_j(v) - d(v,w),$$
$$P_i(w) = P_j(v).$$

Proof. The proof is done by induction upon the radius $k \in [0, a_j(v)]$ of the ball. For $k = 0$, the result is true trivially.

Suppose that the result is true for all vertices in the ball $B(v,k)$, $k \leq a_j(v) - 1$. Now, we consider a vertex w at distance $k+1$ of v. The vertex w has a neighbour u such that $d(v,u) = k$. By induction hypothesis, there exists $i_u \leq j$ such that $a_{i_u}(u) \geq a_j(v) - k$ and $P_{i_u}(u) = P_j(v)$.

Let i be a step, in the steps preceding i_u, where the counter u reached $a_{i_u}(u)$ with $P_i(u) = P_{i_u}(u)$. This step exists for the counter increases of at most 1 at a time, and each time that $P(u)$ is modified, the counter $a(u)$ is reinitialized to -1 (modified rules for GSSP).

Moreover, according to GSSP rule, we have necessarily, $a_i(w) \geq a_{i_u}(u) - 1$ and $P_i(w) = P_i(u)$. Consequently $a_i(w) \geq a_j(v) - k - 1$ and $P_i(w) = P_j(v)$. The result is true for w and so for every vertex at distance $k+1$.

In particular, this proves that at any moment j, for all v, for all $w \in B(v, a_j(v))$, there exists a moment i_w in the past such that $P_{i_w}(w) = P_j(v)$. We now prove that, for all vertices w in the ball $b\frac{a_j(v)}{3}c$, we can choose the same i_w. This is a fundamental property of GSSP algorithm.

Lemma 4.13 (GSSP). Consider an execution of the GSSP algorithm under the hypothesis that the function P is value-convex. For all j, for all v, there exists $i \leq j$ such that for all $w \in B(v, \lfloor \frac{a_j(v)}{3} \rfloor)$, $P_i(w) = P_j(v)$.

Proof.

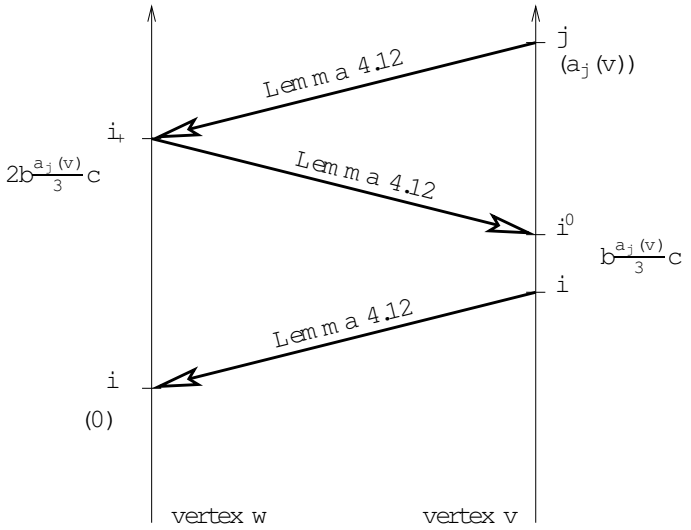


Fig. 5. Proof scheme of Lemma 4.13: vertical axes denote time, the value between brackets are lower bounds for the counter a.

Let i be the first step where $a_i(v) = \lfloor \frac{a_j(v)}{3} \rfloor$ and $P_i(v) = P_j(v)$. Let $w \in B(v, \lfloor \frac{a_j(v)}{3} \rfloor)$, and denote i_+ a step, which existence is given by Lemma 4.12, such that $a_{i_+}(w) \geq a_j v - d(v,w) \geq 2\lfloor \frac{a_j(v)}{3} \rfloor$ and $P_{i_+}(w) = P_j(v)$. Now, let's apply Lemma 4.12 with center w at step i_+. We obtain $i' \leq i_+$ such that $a_{i'}(v) \geq a_{i_+}(w) - d(w,v) \geq \lfloor \frac{a_j(v)}{3} \rfloor = a_i(v)$ and $P_{i'}(v) = P_{i_+}(w) = P_j(v)$. By minimality of i, $i \leq i'$ and finally $i \leq i_+$.

Now, we apply another time Lemma 4.12, with center v, at step i. We obtain then $i' \leq i$, such that $a_{i'}(w) \geq a_i(v) - d(v,w) \geq 0$ and $P_{i'}(w) = P_i(v)$.

To conclude, we obtain two steps i' and i_+ such that $P_{i'}(w) = P_{i_+}(w) = P_j(v)$, and $i' \leq i \leq i_+$. As P is value-convex, we get $P_i(w) = P_j(v)$.

Remark 4.14. One third of the counter is an optimal radius of stability. It is possible to construct examples where the function P is not necessarily constant on the ball of center v and of radius $\lfloor \frac{a_j(v)}{3} \rfloor + 1$.

In these settings, even if the valuation stabilizes, GSSP is always guaranted to not terminate. In order to have noetherian relabellings systems, we define,

given a relabelling system R to which GSSP is applied, $A(R;P;')$ to be the relabelling system based upon R and GSSP with valuation P and adding $'$ to the preconditions of the GSSP Rule. Now termination is closely related to the properties of P and $'$. We define a property, that is only sufficient, for termination.

Definition 4.15. The pair $(P;')$ is uniform if for any run of $A(P;')$, there exist time i and $r_0 \in \mathbb{N}$ such that for all vertex v, for all $j \geq i$, we have $b\frac{a_j(v)}{3}c = r_0 \Rightarrow :'_j(v)$.

Obviously, a uniform pair implies that $A(P;')$ is noetherian. The termination of the increase of $b\frac{a_j(v)}{3}c$ at a node v that first stops, does not prevent the counter at the other nodes to reach this particular value.

4.5 Mazurkiewicz Algorithm + GSSP Algorithm = Universal Local Computation

The main idea in this section is to use the GSSP algorithm in order to compute, in each node, the radius of stability of M. In other words, each node u will know how far other nodes agree with its reconstructed graph $H_{M(u)}$. Let $G = (G;)$ be a labelled graph, let $(G_i)_{0 \leq i}$ be a relabelling chain associated to a run of Mazurkiewicz' Algorithm on the graph G: The vertex v of G_i is associated to the label $((v);(n_i(v);N_i(v);M_i(v)))$: Using the interpretation of section 4.3, this labelling enables to construct a graph that is an asynchronous network snapshot, a would-be cartography of the network.

We now assume the main relabelling system to be M, the valuation to be H. We will have to work a bit on $'$ in order to get a noetherian system. We denote by A_0 the system $A(M;H;false)$. The output of A_0 on the node v is $< H_i(v);a_i(v) >$:

Looking only at the labels, we have, from Theorem 4.1.ii, that H is value-convex. Then, from Lemma 4.13 and Theorem 4.10, we get the main property of the computation of A_0:

Theorem 4.16 (quasi-covering progression). At all step j, for all vertex v, the output of A_0 on v is a couple $< H_j(v);a_j(v) >$ such that if $H_j \neq ?$, then there exists a previous step $i < j$, such that G_i is a quasi-covering of $H_j(v)$ of center v and of radius $b\frac{a_j(v)}{3}c$.

And as the underlying Mazurkiewicz Algorithm is always terminating, we have that the value of H will stabilize with a going to the infinite.

Finally, and considering the previous theorem, we note $r^t(v) = b\frac{a_j(v)}{3}c$, the radius of trust for the algorithm A_0 at node v. In the following section, we show how to get a noetherian relabelling system from A_0.

4.6 Computing Further Informations

The following sections show that M, Mazurkiewicz' algorithm, is actually computing the maximal information we can get distributively. In this subsection, we precise how to extract more information if we have more structural knowledge. In the following, we suppose we know a recursive graph family F to which the underlying network belongs.

Despite the non-recursivity of $\mathbb{P}^{\flat}$ (cf Remark 3.20), we explain how to use the algorithm A and Theorem 4.16 to determine when H is in $\mathbb{P}^{\flat}$. In the remaining of this part, we suppose A_0 is running and all considered H and r are outputs of A_0.

Given these outputs, we describe a (sequential) computation that is done by all the nodes.

```
Data: a graph H ∈ G_L,
r ∈ ℕ.
Result: ? or Yes
repeat
   K ∈ F        /* Enumerate (always in the same order) all the graphs
   of F by order of increasing diameter */
until K is a quasi-covering of center u and radius r of H        /* Loop
ends by Theorem 4.16 */
if K is a quasi-covering of radius r of H for any vertex of H and
r > ρ(H) then
   Output: Yes
else
   Output: ?
end
```

Algorithm 1: $\mathcal{A}_F$: Knowing if H is in $\mathbb{P}^{\flat}$.

We note $\mathcal{A}_F(H,u,r)$ the result of this (semi-)algorithm. If one of the input is clear from the context, it is omitted.

Lemma 4.17. *Suppose a graph family F is given. For any graph H, any vertex u, any $r \in \mathbb{N}$, if $\mathcal{A}_F(H,u,r)$ terminates and outputs Yes, then $H \in \mathbb{P}^{\flat}$.*

Proof. Denote K the quasi-covering that ends the loop of $\mathcal{A}_F$ for input F, H, u, r. As $r \geq \rho(K)+1$, K is a non-strict quasi-covering. K is then a covering of H.

The graph K being in F, we have that $H \in \mathbb{P}^{\flat}$.

Lemma 4.18. *Let $H \in \mathbb{P}^{\flat}$, for all vertices u, v in H, for all r,*

$$\mathcal{A}_F(u,r) = \text{Yes} \iff \mathcal{A}_F(v,r) = \text{Yes}.$$

Proof. Suppose $\mathcal{A}_F(u,r) = \text{Yes}$. Denote K the quasi-covering that ends the loop of $\mathcal{A}_F$ for input u.

By condition for output Yes, we have that K is also a quasi-covering of center v and radius r of H, hence $\mathcal{A}_F(v,r) = \text{Yes}$.

We define $'_I$ to be $\widehat{\mathcal{F}}(H, n(v_0), r^t(v_0)) \neq$ Yes.

We note Carto the system $A(M, H, '_I)$. This will give a noetherian system as demonstrated below. The output of Carto is (H, r^t).

Theorem 4.19 (Asynchronous snapshot). With inputs H and r^t computed by the relabelling system Carto, we have the following properties.

4.19.i The semi-algorithm $\widehat{\mathcal{F}}$ with inputs H and r^t terminates.

4.19.ii At any time, if $\widehat{\mathcal{F}}(H, r^t) =$ Yes, then $H \in \mathcal{P}^{\mathcal{D}}$.

4.19.iii If H is defined, then there exists a previous step i, such that G_i is a quasi-covering of H of center v and of radius r^t.

Proof. The first property is given by Theorem 4.16. The second one is then by Lemma 4.17.

As every run of Carto is a prefix of a run of A_0, we get the final property by 4.16.

Theorem 4.20. The system Carto is noetherian.

Proof. We show that $(H, '_I)$ is uniform. Until M terminates, the modified GSSP part of the system has no significant consequences (the computations of r^t is resetted whenever a rule of M is applied). When M is finished, r^t starts to increase. It will increase until the computation of $\widehat{\mathcal{F}}(H, r^t)$ outputs Yes on some node v.

At this moment, and at this node v, $'_I$ is no more true. As we are working with the final labelling of M, H has the same value on all nodes, hence from Lemma 4.18, the computation of $\widehat{\mathcal{F}}$ will output Yes for the same value of r^t on every node.

Then, Carto is noetherian.

Remark 4.21. As a small optimisation for $\widehat{\mathcal{F}}$, it shall be noticed that it is not to be run if r^t is smaller than the diameter of H.

Without further informations about $\mathcal{F}$, it seems difficult to deduce anything more.

5 Termination Detections

First, we recall from the previous section: let $\mathcal{R}$ be a locally generated relabelling relation, let G a labelled graph, we say that G is an irreducible configuration modulo $\mathcal{R}$ if G is a $\mathcal{R}$-normal form, i.e., no further step with $\mathcal{R}$ is possible ($G \mathcal{R} G'$ holds for no G').

Irreducibility with respect to a relabelling relation yields a notion of implicit termination: the computation has ended – no more relabelling rule can be applied – but no node is aware of the termination. On the other hand, one shall ask a node to be aware of the termination of the algorithm. We will see that one can define various flavour of detection of the termination of a local computation.

- Termination of the algorithm but without detection: implicit termination
- The nodes end in a terminal state, that is a state in which the node knows it will stay forever (whatever is happening elsewhere in the network): detection of the local termination
- The nodes know when all other nodes have computed their final output value. This is the observed termination detection as when termination is detected, some observation computations are not necessarily terminated. Due to the asynchronous aspect of local computations, there is still some observational computations that are going on.
- A node enters a special state that indicates that the algorithm has terminated. This is, obviously, the last step of the computation.

The three last cases are explicit terminations. Termination of a distributed algorithm is usually implicitly assumed to be (one kind of) explicit.

We will see that these various notions are distinct and form a strict hierarchy. First we will give the formal definitions, some examples and then the characterisations of each termination detection. The characterisations are complete except for the local termination detection where we have results only for uniform tasks, that is, local computations ending in a uniform labelling of the network.

5.1 Normalisation of the Labellings

In order to have a unified presentation of the various results, we restrict ourselves to "normalised relabelling systems" w.l.o.g.

Definition 5.1. A normalised labelled graph G is a labelled graph whose labelling is of the form $(mem; out; term)$:

A normalised relabelling system R is a graph relabelling system on normalised graphs where

- mem can be used in preconditions and relabelled,
- out is only relabelled,
- $term$ is only relabelled and has a value in $\{?, Term\}$.

We also use the following convention: if the initial labelled graph is $G = (G, in)$ then it is implicitly extended to the normalised labelling $(G, (in, ?, ?))$. The initial value of mem is therefore given by in.

Now, we make the following assumptions. All graphs are labelled graphs and are all considered to be normalised. All relabelling relations are relabelling relations of normalised labelled graphs.

We also use the following notations. Let G and G' be some given normalised graphs then, for any vertex $u \in G$ (resp. $\in G'$), for any $x \in \{mem, out, term\}$, $x(u)$ (resp. $x'(u)$) is the x component of u in G (resp. G').

The graph x_G is the labelled graph obtained from G by keeping only the x component.

This presentation will find its justifications with the following definitions.

5.2 Termination Detection of Relabelling Systems and of Tasks

In this section, we give the formal definitions of the termination detection for graph relabelling systems. Then we define what is a task, informally it is a relation between the set of inputs and the set of "legal" outputs.

Let F be a given graph family, and R a graph relabelling system. We denote by $Im_R(F)$ the set

$$\{G' \in G_L \mid \exists G \in F, G \xrightarrow{R} G'\}.$$

Implicit Termination There is no detection mechanism. Hence term is not used.

Definition 5.2. A graph relabelling system R has an implicit termination on F if

5.2.i R is noetherian on F,
5.2.ii the term components of any graph in $Im_R(F)$ are all equal to $\top$.

If the underlying local computation is aimed at the computation of a special value, we will, in order to distinguish this value from the intermediate computed values, only look the special purpose component out. As there is no detection of termination, this label is written all over the computation. It becomes significant only when the graph is irreducible, but no node knows when this happens.

Remark 5.3. Such a definition is also relevant for finite self stabilising algorithms [Dol00]. Indeed, one can see implicit termination as a stabilisation. Furthermore, Mazurkiewicz' algorithm has been shown to be self stabilizing [God02].

Local Termination Detection We will now ask the out label to remain unchanged once term is set to Term.

Definition 5.4. A graph relabelling system has a local termination detection (LTD) on F if

5.4.i R is noetherian on F,
5.4.ii term components of graphs in $Irred_R(F)$ are equal to Term,
5.4.iii for all graphs $G, G' \in Im_R(F)$ such that $G \xrightarrow{R} G'$, for every vertex u such that $term(u) = Term$, then

$$out(u) = out'(u),$$
$$term(u) = term'(u) = Term.$$

Remark 5.5. It shall be noted that this definition does not formally prevent a node in a terminal state to act as a gateway by maintaining connectivity of the active parts of the network. Note that the mem component can also be left unchanged with rules that relabel only neighbours. We do not discuss here if, in some sense, a node acting only as gateway is really "terminated". Furthermore, we will only give results for uniform tasks (to be defined later) where these distinctions actually give equivalent definitions.

Observed Termination Detection In this section, we require that once Term appears, all out values have to remain unchanged until the end of the relabellings.

Definition 5.6. A graph relabelling relation R has an observed termination detection (OTD) on F if

- 5.6.i R is noetherian on F,
- 5.6.ii term components of graphs in $\mathrm{Irred}_R(F)$ are equal to Term,
- 5.6.iii for all graphs $G; G' \in \mathrm{Im}_R(F)$ such that $G\ R\ G'$, for all vertex u such that $\mathrm{term}(u) = \mathrm{Term}$, then
  - $\mathrm{term}'(u) = \mathrm{Term}$,
  - for all vertex $v \in G$, $\mathrm{out}(v) = \mathrm{out}'(v)$.

In other words, every node can know when every output value is final. The point is that, in this definition, we ask the network to detect the termination of the computation (in the sense of the out value that is computed), but not to detect the termination of that detection. In the following, we usually have one vertex that detects that the out values are final and then it performs a broadcast of Term. This is actually the termination of this broadcast that we do not ask to be detected. In some sense, this broadcast is performed by an "observer algorithm" whose termination we do not consider.

Remark 5.7. Up to a broadcast, this definition is equivalent to a "weaker" one where it is asked that only at least one vertex of irreducible graphs has a term label set to Term.

Global Termination Detection There is a node that performs explicitly the last relabelling rule.

Definition 5.8. A graph relabelling system R has a global termination detection (GTD) on F if

- 5.8.i R is noetherian on F,
- 5.8.ii for all graphs $G \in \mathrm{Im}_R(F)$, there exists a vertex u such that $\mathrm{term}(u) = \mathrm{Term}$ if and only if G is in $\mathrm{Irred}_R(F)$.

Termination Detection of Tasks We now define tasks by a specification and a domain. The specification is the general description of what we want to do. The domain is the set of labelled graphs where the local computation has to compute the correct outputs with respect to the specification.

First we recall some basic definitions about relations.

Definition 5.9. A relation R is left-total on a set X if for every $x \in X$, there exists y, such that $x R y$.

Definition 5.10. Let a relation R on the set G_L of labelled graphs. Let $X \subseteq G_L$. The restriction of R to X is the relation $R_{|X} = R \cap X \times G_L$:

Definition 5.11. A task T is a couple $(\mathcal{F}, S)$ where S is a (not necessarily locally generated) relabelling relation and $\mathcal{F}$ is a recursive labelled graph family, such that $S_{|\mathcal{F}}$ is left-total on $\mathcal{F}$.

The set $\mathcal{F}$ is the domain of the task, S is the specification of the task.

Note that, in general, a specification S is not particularly related to a given graph family. However, the computability of a task does depend on the domain. See the Election Problem in Sect. 8.5.

A specification can be a decision task such as recognition of property of the underlying graph, or consensus problems, or the problem of election of a node (see Section 8.5), a task depending on the level of structural knowledge we have, the computation of a spanning tree, a d colouration of a graph, etc....

Remark 5.12. It shall be emphasised that we do not explicitly deal with structural knowledge as a parameter for the algorithm. This is exactly the same algorithm that is applied on all the labelled graphs. If there is any parameter to use, this has to be done in the description of the domain and, maybe, encoded in the initial label.

Our definition is aimed at emphasising the difference between the problem – that is the same for any network – and the set of networks on which we want to solve it (if it is solvable at all) with a unique algorithm. E.g., in Section 8.5, we show that, for any minimal graph, there is an Election algorithm but there is no algorithm that solve Election for all minimal graphs.

Keeping in mind the previous remark about structural knowledge, any intuitive task can be encoded by this way.

Example 5.13. We describe the specification of the d colouration problem:

$$\begin{aligned} col_d = \{(G, (G, \gamma)) \mid & \mathrm{card}(\gamma(G)) \leq d, \text{ and} \\ & \forall (u,v) \in E(G),\ \gamma(u) \neq \gamma(v)\}. \end{aligned}$$

A solution to the task $(\mathcal{R}, col_3)$ is presented in Example 3.4.

We now define the computability of a task with respect to the different flavours of termination.

Definition 5.14. A task $(\mathcal{F}, S)$ is locally computable with implicit termination (resp. LTD, OTD, GTD) if there exists a graph relabelling system $\mathcal{R}$ such that

5.14.i (termination) $\mathcal{R}$ has an implicit termination (resp. LTD, OTD, GTD) on $\mathcal{F}$,

5.14.ii (correctness) for any graphs $G \in \mathcal{F}$, $G' \in \mathrm{Irred}_{\mathcal{R}}(G)$

$$G \, S \, out_{G'},$$

5.14.iii (completeness) for any graph $G \in \mathcal{F}$, for any graph G' such that $G \, S \, G'$, there exists G'' such that

$$\begin{aligned} & G'' \in \mathrm{Irred}_{\mathcal{R}}(G), \\ & G' = out_{G''}. \end{aligned}$$

In this case, we say that the graph relabelling relation R computes the task (F, S) with no (resp. local, observed, global) termination detection.

Remark 5.15. The reader should remark that previous definitions ([YK96, BV01]) are restricted to the correctness property. This is the first time, to our knowledge, that the completeness (that can be seen as a kind of fairness property over the legal outputs) is addressed, thus giving its full meaning to the sentence "S is locally computed by R on F".

Moreover, the impossibility results remain true even without the completeness condition (see Remark 6.6).

Remark 5.16. The terms message termination and process termination have also been used to denote implicit and explicit termination [Tel00, introduction for chap. 8].

We denote by $\mathcal{T}_I(F)$ (resp. $\mathcal{T}_{LTD}(F)$, $\mathcal{T}_{OTD}(F)$, $\mathcal{T}_{GTD}(F)$) the set of specifications that are locally computable on domain F with implicit termination (resp. LTD, OTD, GTD). If F is obvious from the context, we will omit it in these notations.

From the definitions, we have

Proposition 5.17. For any labelled graph family F,

$$\mathcal{T}_{GTD}(F) \subset \mathcal{T}_{OTD}(F) \subset \mathcal{T}_{LTD}(F) \subset \mathcal{T}_I(F).$$

Proof. We give the proof, from right to left, as an illustration for those definitions.

A task T with local termination detection has an implicit termination: remove relabelling of term in a relabelling system that computes T with LTD.

Suppose a task T is computable with observed termination detection. A relabelling system that computes T with OTD has LTD by definition.

Suppose now that T is computable with global termination detection by R. An OTD system for T can be obtained by adding a Term-broadcast rule to a relabelling system that computes T with GTD.

Before we characterise these different classes and show that they define a strict hierarchy, we present some examples.

5.3 Four Examples about Computing the Size of an Anonymous Tree

We illustrate these various kinds of termination with the example of the computation of the size of a tree. We give four algorithms: with implicit termination, with local termination detection, with distributed termination detection, with global termination detection. In all cases, we start with the labels of the nodes being uniformly set to $(0, ?, ?)$.

The first three relabelling systems are variations of the fourth one. Thus we focus on the last relabelling system, TreeSize_GTD. The rules are described in their order of appearance.

First we prune the tree starting from the leaves. The size of the pruned subtree is computed incrementally. When the last vertex is pruned, it knows it has the total number of vertices. It broadcasts this value.

When the leaves get the broadcast value, they acknowledge it to their neighbour. Then, the last vertex to get acknowledgements from all its neighbours knows this is the end of the local computation. It shall be noted that this is not necessarily the same pseudo-root vertex at each wave.

We recall that $N(v_0)$ is the set of neighbours of v_0 and that, given a "meta-rule", we enable only rules that modify at least one label. Proofs are left as exercises.

Example 5.18.

TreeSize_I1 : Pruning
Precondition :
$mem(v_0) = 0$,
$\exists! v \in N(v_0); mem(v) = 0$ or $\forall v \in N(v_0); mem(v) \neq 0$.
Relabelling :
$mem'(v_0) = 1 + \sum_{v \in N(v_0)} mem(v)$,
$out'(v_0) = mem'(v_0)$.

TreeSize_I2 : Fast Broadcast
Precondition :
$\forall v \in N(v_0); mem(v) \neq 0$,
Relabelling :
$out'(v_0) = max_{v \in N(v_0)}\{out(v)\}$.

Example 5.19.

TreeSize_LTD1 : Pruning
Precondition :
$mem(v_0) = 0$,
$\exists! v \in N(v_0); mem(v) = 0$.
Relabelling :
$mem'(v_0) = 1 + \sum_{v \in N(v_0)} mem(v)$.

TreeSize_LTD2 : Tree Size is Computed
Precondition :
$mem(v_0) = 0$,
$\forall v \in N(v_0); mem(v) \neq 0$.
Relabelling :
$out'(v_0) = 1 + \sum_{v \in N(v_0)} mem(v)$,
$mem'(v_0) = Size$,
$term'(v_0) = Term$.

TreeSize_LTD3 : Broadcast Size

Precondition :

$\exists v \in N(v_0), mem(v) = Size.$

Relabelling :

$mem'(v_0) = Size.$

$out'(v_0) = max_{v \in N(v_0)}\{out(v)\}.$

$term'(v_0) = Term.$

Example 5.20.

TreeSize_OTD1 : Pruning

Precondition :

$mem(v_0) = 0,$

$\exists! v \in N(v_0), mem(v) = 0.$

Relabelling :

$mem'(v_0) = 1 + \sum_{v \in N(v_0)} mem(v).$

TreeSize_OTD2 : Tree Size is Computed

Precondition :

$mem(v_0) = 0,$

$\forall v \in N(v_0), mem(v) \neq 0.$

Relabelling :

$out'(v_0) = 1 + \sum_{v \in N(v_0)} mem(v),$

$mem'(v_0) = Size.$

TreeSize_OTD3 : Broadcast Size

Precondition :

$\exists v \in N(v_0), mem(v) = Size.$

Relabelling :

$mem'(v_0) = Size,$

$out'(v_0) = max_{v \in N(v_0)}\{out(v)\}.$

TreeSize_OTD4 : End of Broadcast

Precondition :

$card(N(v_0)) \geq 2,$

$mem(v_0) = Size.$

Relabelling :

$mem'(v_0) = Ack.$

TreeSize_OTD5 : Acknowledgement

Precondition :

$\exists v \in N(v_0), mem(v) = Ack.$

Relabelling :

$mem'(v_0) = Ack.$

TreeSize_OTD6 : Termination Detection
Precondition :
$mem(v_0) \neq Ack$,
$\forall v \in N(v_0); mem(v) = Ack$.
Relabelling :
$mem'(v_0) = Term$
$term'(v_0) = Term$.

TreeSize_OTD7 : Broadcast Termination
Precondition :
$\exists v \in N(v_0); mem(v) = Term$.
Relabelling :
$mem'(v_0) = Term$,
$term'(v_0) = Term$.

Example 5.21.

TreeSize_GTD1 : Pruning
Precondition :
$mem(v_0) = 0$,
$\exists! v \in N(v_0); mem(v) = 0$.
Relabelling :
$mem'(v_0) = 1 + \sum_{v \in N(v_0)} mem(v)$.

TreeSize_GTD2 : Tree Size is Computed
Precondition :
$mem(v_0) = 0$,
$\forall v \in N(v_0); mem(v) \neq 0$.
Relabelling :
$out'(v_0) = 1 + \sum_{v \in N(v_0)} mem(v)$,
$mem'(v_0) = Size$.

TreeSize_GTD3 : Broadcast Size
Precondition :
$\exists v \in N(v_0); mem(v) = Size$.
Relabelling :
$mem'(v_0) = Size$.
$out'(v_0) = max_{v \in N(v_0)}\{out(v)\}$.

TreeSize_GTD4 : End of Broadcast
Precondition :
$card(N(v_0)) \quad 2$,
$mem(v_0) = Size$.
Relabelling :
$mem'(v_0) = Ack$.

TreeSize_GTD5 : Acknowledgement

Precondition :

$9v \in N(v_0), mem(v) = Ack.$

Relabelling :

$mem'(v_0) = Ack.$

TreeSize_GTD6 : Termination Detection

Precondition :

$mem(v_0) \neq Ack,$

$\forall v \in N(v_0), mem(v) = Ack.$

Relabelling :

$term'(v_0) = Term.$

5.4 Computing a Spanning Tree

We consider here the problem of building a spanning tree in a graph. We assume that there exists a distinguished vertex, all vertices are initially in some neutral state (encoded by the label ?) except exactly one vertex which is in an active state (encoded by the label ").

The construction of a spanning tree for a rooted network is among the most fundamental tasks to be performed. The spanning tree may be used subsequently for performing broadcast and convergecast communications.

Local Computation of a Spanning Tree With Detection of the Global Termination. The main idea is to use Dewey's prefix-based labelling. The father of the node v is the neighbour labelled by the prefix of v. This encoding is necessary as, here, we restrict to no label on edges or ports. Whenever the covering algorithm is finished, the leaves acknowledge the termination to their fathers until the root node knows everything is over.

The labels mem are words upon the alphabet $\mathbb{N}$. We note $\alpha : \beta$ the concatenation of the words α and β. " denotes the empty word. We define the following notations in order to simplify the description of the rules. Given a vertex v_0, we define $new(v_0) = \{v \in B(v_0) | mem(v) = ?\}$. We also define the set of neighbours labelled by a prefix of the center's label. Let $children(v_0) = \{v \in B(v_0) | mem(v) \in mem(v_0):\mathbb{N}\}$. Given a set X of nodes, we note α_X an injective function $X \to \mathbb{N}$.

The tree has a distinguished vertex, labelled (";?;?), all other nodes are labelled (?;?;?).

Spanning tree1 : Spanning Vertices

Precondition :

$mem(v_0) \neq ?,$

$new(v_0) \neq \emptyset:$

Relabelling :

if $v \in new(v_0), mem'(v) = mem(v_0):\alpha_{new(v_0)}(v):$

Spanning tree2 : Acknowledgement

Precondition :

$mem(v_0) \in \mathbb{N}^+$,

$\forall v \in child(v_0)$, Ack is suffix of $mem(v)$.

Relabelling :

$mem'(v_0) = mem(v)\downarrow Ack$.

Spanning tree3 : GlobalTermination Detection

Precondition :

$mem(v_0) = \varepsilon$,

$\forall v \in child(v_0)$, Ack is suffix of $mem(v)$.

Relabelling :

$term'(v_0) = Term$.

As can be seen from the term label, local termination and global termination are closely related but will differ on the root node. We can note that the nodes know their final number from the first application of a rule (the Spanning Vertices rule), but they do not terminate in order to convergecast the acknowledgement to the root.

6 Characterisations

6.1 Implicit Termination

We need the following definitions to express the local symmetry of a task.

Definition 6.1. A graph family $\mathcal{F}$ is covering-closed if for any graphs G, H such that G is a covering of H, $G \in \mathcal{F} \Rightarrow H \in \mathcal{F}$.

Let $\varphi : G \to H$ be a covering, let H' be a relabelling of H. Then the lifting of H' through φ is the following labelling: $\forall v \in G$, the label of v is the label of $\varphi(v)$ in H'. This labelled graph is denoted $\varphi^{-1}(H')$.

The following proposition is obvious.

Proposition 6.2. Let $\mathcal{F}$ be a graph family. Then $\hat{\mathcal{F}}$ is the smallest graph family containing $\mathcal{F}$ that is covering-closed.

Definition 6.3. Let $\mathcal{F}$ be a covering-closed graph family. A relation R on $\mathcal{F}$ is lifting-closed if for all graphs G and H in $\mathcal{F}$, such that G is a covering of H via φ, for all H', $H\,R\,H' \Rightarrow G\,R\,\varphi^{-1}(H')$.

Definition 6.4. A relabelling relation S is covering-lifting closed on $\mathcal{F}$ if there exists a lifting-closed left-total recursive relation $\hat{S}$ on $\hat{\mathcal{F}}$ such that

$$\hat{S}_{\mid\mathcal{F}} = S_{\mid\mathcal{F}}.$$

Reminding Remark 3.20, we underline that $\underline{P}^{\flat}$, the domain of $\underline{S}^{\flat}$, is not necessarily recursive. Hence we require only that $\underline{S}^{\flat}$ is recursive with left input in $\underline{P}^{\flat}$.

The necessary condition relies upon Lifting Lemma 3.5. This a classical result since the work of Angluin. The sufficient condition uses Mazurkiewicz' algorithm. This result was first proved in a slightly different context in [GMM04]. In [GMM04], the algorithm was quite technically involved. We give here another, maybe simpler, proof using GSSP.

Theorem 6.5. A task (F, S) is locally computable with implicit termination if and only if it is covering-lifting closed.

Proof. Necessary Condition. Let (F, S) be a task that is computable with implicit termination.

There exists R that locally computes (F, S). We define an extension $\underline{S}^{\flat}$ on $\underline{P}^{\flat}$ in the following way: given H in $\underline{P}^{\flat}$, we can apply R until an irreducible form is obtained (this always happens because of Lemma 3.19). We take $H \underline{S}^{\flat} H'$ for any irreducible labelled graph H' obtained from H.

By construction, $\underline{S}^{\flat}$ is left-total on $\underline{P}^{\flat}$. We now show that $\underline{S}^{\flat}$ meets the properties of the covering-lifting closure definition.

First, we show that $\underline{S}^{\flat}$ is lifting-closed. Let H be a labelled graph and $G \in F$ with $\varphi : G \to H$ a covering.

Let H' such that $H \underline{S}^{\flat} H'$. By construction, $H \; R^* \; H'$ and H' is R-irreducible. Hence by the Lifting Lemma 3.5, we have $G \; R^* \; \varphi^{-1}(H')$. Furthermore $\varphi^{-1}(H')$ is irreducible as $\varphi^{-1}(H')$ is. Then $G \underline{S}^{\flat} \varphi^{-1}(H')$.

Finally, we show that the relations $\underline{S}^{\flat}$ and S are equal on F. Let G, G' such that $G \underline{S}^{\flat} G'$. Since R computes S, we have that $G \, S \, G'$.

Let $G \in F$. As R computes locally S on F, for any G' such that $G \, S \, G'$, there exists, by completeness, an execution that leads to an irreducible form equals to G'. Hence $G \underline{S}^{\flat} G'$.

Given the previous result, we get $\underline{S}^{\flat}$ is recursive when the left member is in $\underline{P}^{\flat}$ since S is.

Sufficient Condition. We suppose (F, S) is covering-lifting closed. We will describe a graph relabelling system R^{-1} that computes (F, S).

We first describe a "naive" approach. This approach describes what is essentially at stake here, but, rigorously, it fails for a recursivity reason. This approach is, that at any moment, to take H the computed asynchronous snapshot with M, then choose a H' such that $H \underline{S}^{\flat} H'$ and lift the out labels to the vertices of G. By covering-lifting closure, and by Prop. 4.8, at the end of the computation of M, it will give a correct final labelling. The real problem of this approach is that, in the general case, during the computation, it is not possible to know simply when the computed H is really in $\underline{P}^{\flat}$. Furthermore by Remark 3.20, even knowing F, it is not computable to decide if a given H is in $\underline{P}^{\flat}$.

However, from Th. 4.19.ii, we have a relabelling system C arto that outputs when H is in $\underline{P}^{\flat}$. R^{-1} is obtained by adding to C arto the following rules, for any H' such that $H \underline{S}^{\flat} H'$:

$R^i_{H'}$: Pick an Output
Precondition :
$\widehat{\ }_F(H, n(v_0), r^t(v_0)) = Yes$:
Relabelling :
$choice'(v_0) = H'$.

R^i_{final}: Unifying Outputs
Precondition :
for all $v \in B(v_0)$, $choice(v_0) \preceq choice(v)$:
Relabelling :
for all $v \in B(v_0)$, $choice'(v) = choice(v_0)$:
for all $v \in B(v_0)$, $out'(v) = out_{choice(v_0)}(n(v))$:

The final rule ensures that the same H' is used all over the graph by taking the smallest chosen one.

By Prop. 4.19, R^i is noetherian and the label out is ultimately computed. By covering-lifting closure, the final out labelling is correct. Moreover, by Proposition 4.8 (completeness), we get the completeness condition about S.

Remark 6.6. If we drop the completeness property from the requirement, the proof shows that it is only necessary and sufficient to have $S^{\flat}_{\mathcal{F}} \subseteq S_{\mathcal{F}}$.

Remark 6.7. If it is easy (read recursive) to check whether a given graph is in $\mathcal{F}^{\flat}$ – for example if $\mathcal{F}$ is covering-closed – the algorithm above is very much simplified because the main difficulty is to know when a "Pick an output" rule can be applied. This reveals to be actually the case for almost all practical cases.

6.2 Local Termination Detection of Uniform Tasks

The results of this part comes from [GM 03]. They stand only for uniform tasks, that is, for tasks with a uniform out label. We adapt the definitions to the context of this paper and we give the main result. The complete proofs (that are basically the same up to the notations) and some applications, in particular about the problem of deducing by local computations a structural information from another one, are given in [GM 03].

Definition 6.8. A task is uniform if for every $G \in \mathcal{F}$, every G' such that $G \subseteq G'$, every vertices $u, v \in G$, $out_{G'}(u) = out_{G'}(v)$. In this case, the task is denoted by $(\mathcal{F}, f)$ where $f : \mathcal{F} \to L$ is the final labelling function.

Definition 6.9. A uniform task $(\mathcal{F}, f)$ is quasi-covering-lifting closed if there exists a recursive function $r : \mathcal{F}^{\flat} \to \mathbb{N}$ such that, if there exist graphs K, K' in $\mathcal{F}$ and H such that K and K' are quasi-coverings of H of radius $r(K)$, then $f(K) = f(K')$.

Theorem 6.10 ([GM 03]). A uniform task is locally computable with local termination detection if and only if it is quasi-covering-lifting closed.

6.3 Observed Termination Detection

Theorem 6.11. A task $T = (F, S)$ is locally computable with observed termination detection if and only if

6.11.i T is covering-lifting closed,

6.11.ii there exists a recursive function $r : \mathbb{P}^{fb} \to \mathbb{N}$ such that for any $H \in \mathbb{P}^{fb}$, there is no strict quasi-covering of H of radius $r(H)$ in F.

Proof. Necessary Condition. This is actually a simple corollary of the quasi-lifting lemma.

We prove this by contradiction. We assume there is a graph relabelling system R with observed termination detection that computes the specification S on F.

Now we suppose there exists $H \in \mathbb{P}^{fb}$ that admits strict quasi-coverings of unbounded radius in F. By Lemma 3.19, R is noetherian for H. Consider an execution of R of length l.

By hypothesis, there exists $K \in F$ a strict quasi-covering of H of radius $2l+1$. By the quasi-lifting lemma, we can simulate on a ball of radius $2l+1$ of K the execution of R on H. At the end of this relabelling steps, there is a node in K that is labelled Term. As the quasi-covering K is strict, there exists at least one node outside of the ball that has not even taken a relabelling step of R, hence that has not written anything to out. Hence R has not the observed termination property on K. A contradiction.

Sufficient Condition. In some sense, we will observe the termination of R^i by letting r^t increase a bit more. In order to do that, we have to relax the condition φ_I.

We define the condition φ_O by[2]:

- $\widehat{F}(H, n(v_0), r^t(v_0)) \in$ Yes or $r^t(v_0) \geq r(H)$;
- $\widehat{F}(H, n(v_0), r^t(v_0)) \in$ Yes or $out(v_0) = out_{choice(v_0)}(n(v_0))$.

In order to define R^O, we add to $A((H, choice), \varphi_O)$ the following rule:

R^O-H': Termination Detection and Pick an Output

Precondition :

$\widehat{F}(H, n(v_0), r^t(v_0)) =$ Yes.

Relabelling :

$choice'(v_0) = H'$.

R^O: Unifying Outputs

Precondition :

for all $v \in B(v_0)$, $choice(v_0) \geq choice(v)$.

Relabelling :

for all $v \in B(v_0)$, $choice'(v) = choice(v_0)$.

for all $v \in B(v_0)$, $out'(v) = out_{choice(v_0)}(n(v))$.

[2] with the convention that – in order to avoid the problems of definition of H, or its belonging to $\widehat{F}$ – in the or conditions, the right part is not "evaluated" if the left part is true.

R° : Termination Detection

Precondition :

$\hat{r}^t(v_0) > r(H)$:

Relabelling :

$term^0(v_0) = Term$:

This system computes the task (F, S) and has an observed termination detection.

First, R° is noetherian. The valuation is now slightly different of the one of Carto, but we can use the same proof as for Theorem 4.20 to prove that $((H, choice), \prime_{\circ})$ is uniform. Here again, the GSSP Rule will stop being enabled on each vertex for the same value of r^t, the one that is equal to $r(H) + 1$.

Now, suppose we have, at a given time i, on a node v, $r(H(v)) < r^t(v)$, then, by the hypothesis 6.11.ii and by the Remark 3.10, the entire graph G_i is a covering of H. Hence M is terminated. Furthermore, the second precondition of $\prime_{\circ}$ ask out to be computed on each vertex, from the same graph H as choice is a component of the valuation.

Thus the detection of termination is correct. Moreover, by covering-closure, the out labels are correct for the specification S.

In the following we refer to hypothesis 6.11.ii as the relatively bounded radius of quasi-covering condition.

6.4 Global Termination Detection

In this section, we characterise the most demanding termination mode.

Theorem 6.12. A task (F, S) is locally computable with global termination detection if and only if

- any labelled graph in F is covering minimal,
- there exists a recursive function $r : F \to \mathbb{N}$ such that for any $G \in F$, there is no quasi-covering of G of radius $r(G)$ in F, except G.

Proof. Necessary Condition. We need only to prove the first item. Asminimality implies $F = \hat{P}^{\flat}$, the second one is a restatement of the one for termination detection by observer.

The minimality of any graph in F is again a corollary of the lifting lemma. Suppose there are G and H in F such that G is a strict covering of H.

We consider a relabelling chain in H. It comes from the lifting lemma that this can be lifted step by step in G. When the final step is reached in H, and as G is a strict covering of H, there are at least two nodes in G where to apply the finalTerm rule. Hence a contradiction.

Sufficient Condition. The two hypothesis imply that task (F, S) has the observed termination detection property (the covering-lifting property is a trivial tautology when all concerned graphs are minimal). Hence there exists a relabelling system R° that computes S with OTD.

We define R^g the relabelling system obtained by the union of R° without the "Termination Detection" rules and the rules of Section 5.4 for the computation of a spanning tree. The root is the vertex that gets number 1 in M, when this vertex observes the termination for R° with the following rule:

R^g : Root

Precondition :

$\dot{r}(v_0) > r(H)$,

$n(v_0) = 1$.

Relabelling :

$mem'(v_0) = \varepsilon$.

By minimality of G, there is only one vertex with number 1 when M is finished. Hence we really get a spanning tree and not a spanning forest.

7 Applications

In this section, we present consequences from the previous theorems. There are known computability results, some new ones and the proof that the different notions of termination detection are not equivalent.

We emphasise that the following results are bound to the model of local computations. Results on other models should be similar even if strictly and combinatorially speaking different. They remain to be precisely described and computed.

7.1 Domains and Specifications

Consider a locally computable task $T = (F, S)$. The first remark is that implicit termination and LTD give conditions on the specification (with respect to the domain) but there are (sometimes trivial) tasks on any domain. And on the contrary, OTD and GTD have conditions upon the domain and (weak) conditions on the specification. The difference between domains that have OTD for (almost) any task and the ones that have only GTD for any tasks depends upon the covering-minimality of the graphs in the given domain.

As a conclusion, with respect to the termination detection criteria, whether we can work where we want but we cannot do what we want (the specification has to respect covering-lifting and quasi-covering-lifting closures), whether we can do whatever we want, but we can do it only on particular families of networks. The more interesting possible trade-off is probably on the LTD tasks but that is the most complex families and its complete characterisation has yet to be done.

7.2 Known Results as Corollaries

We first sum up some results for every category of termination detection. Then we show that the hierarchy is strict. With the remark from the previous subsection, we focus mainly on the relevant part (domain or specification). A very important application for distributed algorithms, the Election problem, is done in a dedicated section, Section 8.

Implicit Termination. From Th.6.5, we can see that what can be computed with implicit termination depends only of what is kept whenever there is lifting. Such a property is called "degree-refinement" in the graph-theoretic context [Lei82]. Hence, what can be computed with implicit termination is exactly a computation about the degree-refinement of the network. See [GMM04] about an investigation of the decision task of recognising whether the underlying network belongs to a given class.

Example 7.1. We denote by $\mathcal{R}$ the family of rings. Consider the following task $T_1 = (\mathcal{G}, \simeq_{\mathcal{R}})$ which asks to decide whether the network is a ring or not. The task T_1 is locally computable with implicit termination but not with a relabelling system with LTD. Consider chains. Long ones are quasi-coverings of arbitrary radius for a given ring. Hence $\simeq_{\mathcal{R}}$ is not quasi-covering-lifting closed.

We give a second example with domain $\mathcal{R}$.

Example 7.2. We denote Div the following specification: the out labels are taken in $\mathbb{N}$ and $G\ Div\ G'$ if and only if the final out label divides the size of G.

$(\mathcal{R}, Div)$ is covering-lifting closed as a ring G is a covering of a ring H if and only if the size of G divides the size of H. However, Div is not quasi-covering-lifting closed. There are "huge" minimal rings that are quasi-covering of arbitrary radius of, say, R_7.

Local Termination Detection. See [GM03] for numerous examples about of the computation of a structural knowledge (that is a uniform labelling) from another one.

Example 7.3. The relation $Colo_3$ is the specification of the 3-colouring problem. The task $T_2 = (\mathcal{R}, Colo_3)$ has local termination detection (relabelling system given in Example 3.4) but has not observed termination detection for there are "huge" rings that are quasi-covering of any given arbitrary radius of, say, R_3.

Observed Termination Detection. Here we will find the frequent (sometimes implicit) assumptions usually made by distributed algorithms:

- size or diameter is known,
- a bound on the diameter or the size is known.

It shall be noted that the computability results from the work of Yamashita and Kameda belong to this category.

Example 7.4. Let $n \in \mathbb{N}; n \geq 6$. We note $\mathcal{R}^n$ the rings of size at most n. We consider $T_3 = (\mathcal{R}^n; Colo_3)$. The radius of strict quasi-covering are bounded in $\mathcal{R}^n$. Hence T_3 has OTD, but it has not GTD, for the ring R_6 is not covering-minimal.

Global Termination Detection Here we really find all the well known assumptions usually made about distributed network algorithms. The theorems admit well known corollaries; more precisely from Theorem 6.12 we deduce immediately that we have global termination detection for any task for the following families of graphs:

- graphs having a leader,
- graphs such that each node is identified by a unique name,
- trees.

From Theorem 6.12 we deduce there is no such termination for:

- the family of covering-minimal anonymous rings,
- the family of covering-minimal anonymous networks.

Example 7.5. Let $n \in \mathbb{N}$. We note $\mathcal{PR}^n$ the rings of prime size at most n. We consider $T_4 = (Colo_3; \mathcal{PR}^n)$. The radius of quasi covering are bounded in $\mathcal{PR}^n$, and rings of prime size are covering-minimal. Hence T_4 is in T_{GTD}.

7.3 The Hierarchy is Strict

The previous examples T_1, T_2 and T_3 show that the hierarchy is strict and that the four notions of termination are different.

Proposition 7.6.

$$T_{GTD}(G) = T_{OTD}(G) = \emptyset \subset T_{LTD}(G) \subset T_I(G);$$
$$T_{GTD}(R) = T_{OTD}(R) = \emptyset \subset T_{LTD}(R) \subset T_I(R);$$
$$T_{GTD}(\mathcal{R}^n) = \emptyset \subset T_{OTD}(\mathcal{R}^n).$$

7.4 New Corollaries

New interesting corollaries are obtained from these theorems.

Multiple leaders From Theorem 6.11 and Lemma 3.12, we get

Corollary 7.7. Any covering-lifting closed task has an OTD solution in the following families:

- graphs having exactly k leaders,
- graphs having at least one and at most k leaders.

From Theorem 6.11, we deduce a negative result for the family of graphs having at least $k \geq 2$ leaders.

Link Labellings and Sense of Direction. We recall that a homomorphism φ from the labelled graph G to the labelled graph G' is a graph homomorphism from G to G' which preserves the labelling: a node is mapped to a node with the same label and a link is mapped to a link with the same label.

Thus, a family of labelled graphs induced by a weak sense of direction satisfies the condition 6.11.ii of Theorem 6.11 (indeed weak sense of direction forbids quasi-coverings). Thus, for any task, observed termination detection is possible in all families of graphs with weak sense of direction.

7.5 About the Complexity of Local Computations

The step complexity of M is $O(n^3)$ [God02]. Denote C the complexity of GSSP in the bounded radius of quasi-covering context. Hence we can see that the complexity of a task is bounded by $O(n^2 + C)$. It is easy to see that the complexity of GSSP is closely related to the bound r of the radius of quasi-coverings. When M is terminated, any node has to go from 0 to r with GSSP rule. Thus $C \quad (r+1)$.

Whether the complexity comes from the distributed gathering of information or from the termination detection depends upon the order of magnitude of r.

A similar study of the complexity of distributed algorithms by upper-bounding by "universal algorithm" is done in [BV02b] where, it shall be noted, the notion of quasi-covering is introduced for trees.

8 A Characterisation of Families of Networks in which Election is Possible

Considering a labelled graph, we say informally that a given vertex v has been elected when the graph is in a global state such that exactly one vertex has the label `Elect` and all other vertices have the label `Non-Elect`. The labels `Elect` and `Non-Elect` are terminal, i.e., when they appear on a vertex they remain until the end of the computation. This is the standard definition.

Note that if we ask nothing about the non elected vertices, this gives an equivalent definition in terms of computability. Because when a node is elected, it can broadcast it to all the nodes of the networks.

Definition 8.1. Let F be a class of connected labelled graphs. Let R be a locally generated relabelling relation, we say that R is an election algorithm for the class F if R is noetherian and for any graph G of F and for any normal form G' obtained from G; $G \; R \; G'$; there exists exactly one vertex with the label `Elect` and all other vertices have the label `Non-Elect`.

With the notation of the previous part, we have the various definitions for the various kinds of termination detection.

Definition 8.2. Let F be a class of connected labelled graphs. Let `Election` be the following relation: G and G' are in relation if and only if there exists in

G' exactly one vertex with the label Elect and all other vertices have the label Non-Elect.

The implicit (resp. LTD, OTD, GTD)-Election on F is the task $(F, \text{Election})$ with implicit (resp. local, observed, global) termination detection.

We underline that we are looking for classes of networks that admit the same Election algorithm for all its elements. Having an algorithm that works for several networks (say, independently of the knowledge of its size) is very important for reliability. In this setting, saying that G admits an Election algorithm amounts to say that $(\{G\}, \text{Election})$ is a computable task. It is important to note that saying that Election is computable on a given family F does not mean that $(\{G\}, \text{Election})$ is a computable task for any $G \in F$, but means that $(F, \text{Election})$ is a computable task.

We can see that the definition of LTD-Election is equivalent to the standard definition of Election.

We will prove that the possibility of the LTD-Election on F is equivalent to the possibility of the GTD-Election. But first we give two examples of elections.

8.1 Two Examples

An Election Algorithm in the Family of Anonymous Trees. The following relabelling system elects in trees. The set of labels is $L = \{N, \text{Elect}, \text{Non-Elect}\}$. The initial label on all vertices is N.

Election_Tree1 : Pruning rule

Precondition :

$\lambda(v) = N$,

$\exists! v \in B(v, 1); v \neq v_0;\ \lambda(v) = N$.

Relabelling :

$\lambda'(v_0) = \text{Non-Elect}$:

Election_Tree2 : Election rule

Precondition :

$\lambda(v) = N$,

$\forall v \in B(v, 1); v \neq v_0;\ \lambda(v) \neq N$.

Relabelling :

$\lambda'(v_0) = \text{Elect}$:

Let us call a pendant vertex any vertex labelled N having exactly one neighbour with the label N: There are two meta-rules Election_Tree1 and Election_Tree2: The meta-rule Election_Tree1 consists in cutting a pendant vertex by giving it the label Non-Elect: The label N of a vertex v becomes Elect by the meta-rule Election_Tree2 if the vertex v has no neighbour labelled N: A complete proof of this system may be found in [LMS99].

An Election Algorithm in the Family of Complete Graphs. The following relabelling system elects in complete graphs. The set of labels is $L = \{N, Elect, Non\text{-}Elect\}$. The initial label on all vertices is $l_0 = N$.

Election_Complete-graph1 : Erasing rule
Precondition :
$mem(v_0) = N$,
$\exists v \in B(v_0, 1), v \neq v_0, mem(v) = N$.
Relabelling :
$mem'(v_0) = Non\text{-}Elect$.

Election_Complete-graph2 : Election rule
Precondition :
$mem(v_0) = N$,
$\forall v \in B(v_0, 1), v \neq v_0, mem(v) \neq N$.
Relabelling :
$mem'(v_0) = Elect$.

It is straightforward to verify that this system elects in the family of complete graphs.

8.2 Characterisation of Election

We show that the LTD-Election is solvable if and only if the GTD-Election is solvable. Then we use the general characterisation of this paper to conclude.

Proposition 8.3. Let F be a labelled graph family. The LTD-Election task on F is computable if and only if the GTD-Election is.

Proof. The sufficient condition is easy (Proposition 5.17). We focus on the necessary condition.

Suppose R is a graph relabelling relation with LTD solving the Election task on F. In order to convert it in a graph relabelling relation with GTD, we will add some rules to R. We add a rule that starts the computation, with GTD, of a spanning tree rooted in the Elect vertex. This standard construction is given in Section 5.4.

Remark 8.4. This demonstration shows that even if we define a task with a LTD flavour, it can reveal to be in the GTD family of tasks because of the form of the specification. Furthermore, we will now not distinguished between LTD (resp. OTD, GTD)-Election.

As a corollary of Theorem 6.12, we get:

Theorem 8.5. Let F be a class of connected labelled graphs. There exists an Election algorithm for F if and only if

- graphs of F are minimal for the covering relation, and
- there exists a computable function $r : F \to \mathbb{N}$ such that for all graph G of F, there is no quasi-covering of G of radius greater than $r(G)$ in F, except G itself.

Remark 8.6. In fact, the Election algorithm can be directly derived from the Carto algorithm. When a node detects the termination of M, it sets its out label to Elect or Non-Elect whether it is numbered 1 or not.

8.3 Applications

The first attempt of a complete characterisation of election was first done in [BCG+96], but the results were only given when a bound upon the diameter is initially known. In the general no knowledge case, they give a "pseudo"-election algorithm, i.e., some Elect labels can appears during the computation, this is only when the computation is finished that this label has to be unique. This is exactly the definition of implicit-Election.

Known results appear now as simple corollaries of Theorem 8.5.

- [Maz97] Covering minimal networks where the size is known;
- Trees, complete graphs, grids, networks with identities.

Those last families contains no q-sheeted quasi-covering of a given graph for $q \geq 2$, hence the r function can be twice the size of the graph, see Lemma 3.12.

We also get some new results. An interesting result is that there is no election algorithm for the family of all the networks where the election is possible.

Proposition 8.7. Let G be a labelled graph. Election is computable on G if and only if G is covering-minimal.

Proposition 8.8. There is no Election algorithm on the family of covering-minimal graphs.

Proof. Rings with a prime size are minimal and does not respect the relatively bounded quasi-covering condition.

However, from Theorem 6.5, it is easy to derive where implicit-Election is computable.

Proposition 8.9. Election is computable with implicit termination on the family of covering-minimal graphs.

We obtained as a direct corollary:

Proposition 8.10. There exists an election algorithm for covering minimal graphs where a bound of the size is known.

We can notice that no trivial extension of the proof of the Mazurkiewicz algorithm enables to obtain directly this proposition.

We also have a new and interesting result for graphs with at most k distinguished vertices:

Proposition 8.11. Let $k \in \mathbb{N}$. Let $\mathcal{I}$ be a family of covering-minimal $\{0;1\}$-labelled graphs such that for all graph, there are at most k vertices labelled with 1. Then, there exists an election algorithm for this family.

Proof. We define $r(G) = (k+1)|V(G)|$ and we remark that quasi-covering in $\mathcal{I}$ can be at most k-sheeted. Hence, by Lemma 3.12, we deduce that r has the desired property.

From this proposition we deduce that to have an Election algorithm in a network where uniqueness of an identity is not guaranteed, we only need a bound on the multiplicity of identities.

9 Conclusion

9.1 Characterisations of termination detection

Distributed algorithms are very different from sequential ones. How to make them terminate is a difficult problem. Moreover in this paper, we show that even if the termination is given, and so can be detected by an omniscient observer, the detection of this fact is not always possible for the nodes inside the network.

In this paper, we present a quite comprehensive description of the computability of tasks with explicit detection of the termination. We show one can define four flavours of termination detection: implicit termination detection, local termination detection, termination detection by a distributed observer and global termination detection. For each termination detection, we give the characterisations of distributed tasks that admit such a termination detection, and we show they form a strict hierarchy. The local termination detection flavour is only characterised in the case of uniform tasks. It has yet to be completely investigated.

We prove that if we ask for implicit or local termination detection, we can work in any family of networks, but the computable tasks are restricted. On the other hand, we show that if we ask for global termination detection, we have to work on special classes of graphs – minimal graphs with relatively bounded radius of quasi-coverings – but there, every task is computable. This characterisation precisely explains numerous kind of hypothesis that are traditionally made when designing distributed algorithms.

In conclusion, we show that a distributed task is not only described by a specification – a relation between inputs and outputs –, a domain – the family of networks in which we have to meet the specification –, but also by the kind of termination detection we ask for.

9.2 Comparison with other models

In contrast with previous works about the computability of distributed tasks, we can say that, usually, the termination of the distributed algorithms is "factored out": the nodes know at the beginning an upper bound on the number of steps

it will take. For Yamashita and Kameda models and Boldi and Vigna models, it is the particular initial knowledge that enable to determine how many steps of union of local views is sufficient.

It can be observed that, actually, the universal algorithms in these works are constituted by a potentially infinite loop (merge local views for Yamashita/Kameda and Boldi/Vigna, snapshot read-write for Herlihy/Shavit and Borowsky/Gafni [BG93]) and an external condition to say when to end the infinite loop. This condition does not depend on the distributed computations. In this sense, we can say that the termination is factored out: it is not detected in a truly distributed way as the number of rounds is known in advance, it does not depend of what is gathered by each node in the exchange of information of the distributed algorithm.

In a kind of contrast, we can see that our asynchronous snapshot algorithm is constituted of two parts: Mazurkiewicz' algorithm, that is always terminating (implicit termination); and the generalised SSP stability detection that does not terminate alone. That is this combination that enables to detect, in a truly distributed way, the termination of the distributed tasks. When to stop GSSP is computed from the value obtained by Mazurkiewicz' algorithm, and not from a given a priori value like in the other approachs.

9.3 Impossibility results in non-faulty networks

The results given in this paper show that there are also possibility/impossibility results even with non-faulty networks. This paradox could be settled in the recent approach of failure detectors: the various kind of distributed systems can be seen as a perfect system (synchronous, centralised, with identities ...) with various failure (asynchronicity, node failures, communication failures, ...) [CT96, Gaf98]. In this contribution, we show that lack of structural knowledge (nodes do not know exactly what is the topology of their network), and lack of structural information (e.g. unique identities) are also a kind of failure in this concern.

The authors wish to thank the anonymous referees for some helpful comments. They are also specially grateful to Bruno Courcelle, Pierre Casteran and Vincent Filou for their corrections and stimulating questions regarding the previous version of this report.

References

Ang80. D. Angluin. Local and global properties in networks of processors. In Proceedings of the 12th Symposium on Theory of Computing, pages 82–93, 1980.

AW04. H. Attiya and J. Welch. Distributed computing: fundamentals, simulations, and advanced topics. McGraw-Hill, 2004.

BCG+96. P. Boldi, B. Codenotti, P. Gemmell, S. Shammah, J. Simon, and S. Vigna. Symmetry breaking in anonymous networks: Characterizations. In Proc. 4th Israeli Symposium on Theory of Computing and Systems, pages 16–26. IEEE Press, 1996.

BG93. E. Borowsky and E. Gafni. Immediate atomic snapshots and fast renaming. In Proc. of the 12th Annual ACM Symposium on Principles of Distributed Computing, 1993.

BL86. H.-L. Bodlaender and J. Van Leeuwen. Simulation of large networks on smaller networks. Information and Control, 71:143–180, 1986.

Bod89. H.-L. Bodlaender. The classification of coverings of processor networks. J. Parallel Distrib. Comput., 6:166–182, 1989.

BV99. P. Boldi and S. Vigna. Computing anonymously with arbitrary knowledge. In Proceedings of the 18th ACM Symposium on Principles Of Distributed Computing, pages 181–188. ACM Press, 1999.

BV01. P. Boldi and S. Vigna. An effective characterization of computability in anonymous networks. In Distributed Computing. 15th International Conference, DISC 2001, volume 2180 of Lecture notes in computer science, pages 33–47. Spinger-Verlag, 2001.

BV02a. P. Boldi and S. Vigna. Fibrations of graphs. Discrete Math., 243, 2002.

BV02b. P. Boldi and S. Vigna. Holographic trees. In LATIN '02, number 2286 in Lecture Notes in Computer Science, pages 465–478, Cancun, Mexico, 2002. Springer Verlag.

BV02c. P. Boldi and S. Vigna. Universal dynamic synchronous self-stabilization. Distr. Computing, 15(3), 2002.

CGM08. J. Chalopin, E. Godard, and Y. Metivier. Distributed algorithms with local termination. in preparation, 2008.

CGMT07. J. Chalopin, E. Godard, Y. Metivier, and G. Tel. About the termination detection in the asynchronous message passing model. In Proc. of SOFSEM 2007: Theory and Practice of Computer Science, volume 4362 of Lecture Notes in Computer Science, pages 200–211, 2007.

CM07. J. Chalopin and Y. Metivier. An efficient message passing election algorithm based on Mazurkiewicz's algorithm. Fundamenta Informaticae, 80(1-3):221–246, 2007.

CT96. T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. Journal of the ACM, 43(2):225–267, March 1996.

Dol00. S. Dolev. Self-Stabilization. MIT Press, 2000.

Gaf98. E. Gafni. Round-by-round fault detectors (extended abstract): unifying synchrony and asynchrony. In Proc. of PODC '98, pages 143–152. ACM Press, New York, 1998.

GM02. E. Godard and Y. Metivier. A characterization of families of graphs in which election is possible (ext. abstract). In M. Nielsen and U. Engberg, editors, Proc. of Foundations of Software Science and Computation Structures, FOSSACS'02, number 2303 in LNCS, pages 159–171. Springer-Verlag, 2002.

GM03. E. Godard and Y. Metivier. Deducible and equivalent structural knowledges in distributed algorithms. Theory of Computing Systems, 36(6):631–654, 2003.

GMM04. E. Godard, Y. Metivier, and A. Muscholl. Characterization of Classes of Graphs Recognizable by Local Computations. Theory of Computing Systems, 37(2), 2004.

God02. E. Godard. A self-stabilizing enumeration algorithm. Information Processing Letters, 82(6):299–305, 2002.

IR81. A. Itai and M. Rodeh. Symmetry breaking in distributive networks. In Proceedings of the 13th Symposium on theory of computing, pages 150–158, 1981.

KY96. 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.

Lei82. F. T. Leighton. Finite common coverings of graphs. J. Combin. Theory, Ser. B, 33:231–238, 1982.

LeL77. G. LeLann. Distributed systems: Towards a formal approach. In B. Gilchrist, editor, Information processing'77, pages 155–160. North-Holland, 1977.

LL90. L. Lamport and N. Lynch. Distributed computing: models and methods. Handbook of theoretical computer science, B:1157–1199, 1990.

LMS99. I. Litovsky, Y. Metivier, 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.

LMZ95. I. Litovsky, Y. Metivier, and W. Zielonka. On the Recognition of Families of Graphs with Local Computations. Information and Computation, 118(1):110–119, 1995.

Lyn96. N.A. Lynch. Distributed algorithms. Morgan Kaufman, 1996.

Mas91. W.S. Massey. A basic course in algebraic topology. Springer-Verlag, 1991. Graduate texts in mathematics.

Mat87. F. Mattern. Algorithms for distributed termination detection. Distributed computing, 2:161–175, 1987.

Maz87. 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.

Maz88. A. Mazurkiewicz. Solvability of the asynchronous ranking problem. Inf. Processing Letters, 28:221–224, 1988.

Maz97. A. Mazurkiewicz. Distributed enumeration. Inf. Processing Letters, 61:233–239, 1997.

MMW97. Y. Metivier, A. Muscholl, and P.-A. Wacrenier. About the local detection of termination of local computations in graphs. In D. Krizanc and P. Widmayer, editors, SIROCCO 97 – 4th International Colloquium on Structural Information & Communication Complexity, Proceedings in Informatics, pages 188–200. Carleton Scientific, 1997.

MT00. Y. Metivier and G. Tel. Termination detection and universal graph reconstruction. In SIROCCO 00 – 7th International Colloquium on Structural Information & Communication Complexity, pages 237–251, 2000.

NS95. M. Naor and L. Stockmeyer. What can be computed locally? SIAM Journal on Computing, 24(6):1259–1277, 1995.

Rei32. K. Reidemeister. Einfuhrung in die Kombinatorische Topologie. Vieweg, Brunswick, 1932.

RFH72. 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.

Ros00. K.H. Rosen, editor. Handbook of discrete and combinatorial mathematics. CRC Press, 2000.

SSP85. B. Szymanski, Y. Shy, and N. Prywes. Synchronized distributed termination. IEEE Trans. Software Eng., 11(10):1136–1140, 1985.

Tel00. G. Tel. Introduction to distributed algorithms. Cambridge University Press, 2000.

YK96. M. Yamashita and T. Kameda. Computing on anonymous networks: Part i – characterizing the solvable cases. IEEE Transactions on parallel and distributed systems, 7(1):69–89, 1996.