ELSEVIER

Information Processing Letters 82 (2002) 299–305

Information Processing Letters

www.elsevier.com/locate/ipl

# A self-stabilizing enumeration algorithm

Emmanuel Godard

*LaBRI U.M.R. 5800, Université Bordeaux I, 351 Cours de la Libération, 33405 Talence Cedex, France*

Received 25 April 2001; received in revised form 21 September 2001
Communicated by L. Boasson

**Abstract**

This paper describes a self-stabilizing version of an algorithm presented by A. Mazurkiewicz [Inform. Process. Lett. 61 (1997) 233–239] for enumerating nodes by local rules on an anonymous network. The result improves the reliability aspects of the original algorithm and underlines the importance of a non-ambiguous topology for a network. 

*Keywords:* Distributed computing; Self-stabilizing systems; Local computations; Anonymous networks

## 1. Introduction

Dijkstra first introduced the concept of self-stabilizing algorithms in [1] with the view to design algorithms with a demanding reliability; starting in any corrupted initial state, these algorithms achieve the sought goal in finite time without centralized intervention. The theory of anonymous networks describes the solvability or non-solvability of different problems of global conflicts by local computations without a unique identity for each node being known. An enumeration algorithm is a distributed solution to the problem of affecting to each node of an initially anonymous network a different *name*, names are taken in the set $\{1, 2, \ldots, n\}$ ($n$ is the size of the network). This paper describes a self-stabilizing solution to the Enumeration Problem.

Mazurkiewicz shows the enumeration problem was only solvable in so-called non-ambiguous graphs, and presents his algorithmic solution [5]. It is worth noting that this algorithm, despite its simplicity, provides all the underlying graph-related information that a distributed algorithm can be required to compute [2]. In this paper, we show how to modify the original algorithm in order to make it work properly even starting from a corrupted state.

The existence of such a self-stabilizing algorithm demonstrates that non-ambiguous topology is an interesting topology for designing very robust distributed algorithms that can recover from all kind of failures. It is thus not only of theoretical interest to determine what works in this very worst case scenario (unavailability of helpful identities and arbitrary initial values), it has also some important robustness consideration since the family where the problem is solvable is quite large.

This paper is organized as follows, first we present some notations and definitions that are essentially standard and we restate the definitions of [5]. We describe the self-stabilizing enumeration algorithm and prove its correctness: the main observation being that the algorithm terminates with a bijective labeling

*E-mail address:* godard@labri.fr (E. Godard).

0020-0190/01/$ – see front matter 
PII: S0020-0190(01)00291-5

whose conversion to an enumeration is then a technical question. We also give bounds on the time of execution that were not given in the original paper.

## 2. Basic notions and notations

### 2.1. Graphs

The notation used here is essentially standard. A graph $G$ is defined as a finite set $V(G)$ of vertices together with a set $E(G) \subseteq \binom{V}{2}$ of edges. We only consider finite, undirected graphs without multiple edges and self-loops. Let $e = \{v, v'\}$ be an edge: we say that $e$ is incident with $v$, and that $v$ is a neighbor of $v'$. The set of neighbors of a vertex $v$ is denoted $N_G(v)$. Let also $B_G(v) = N_G(v) \cup \{v\}$.

A path $P$ from $v_1$ to $v_i$ in $G$ is a sequence $P = (v_1, e_1, v_2, e_2, \ldots, e_{i-1}, v_i)$ of vertices and edges such that for all $1 \leqslant j < i$, $e_j$ is an edge incident with the vertices $v_j$ and $v_{j+1}$; the integer $i - 1$ is the length of $P$. The distance between two vertices is the length of the shortest path between those vertices. Let $v$ be a vertex, $k$ an integer, we denote by $B_G(v, k)$ the ball of center $v$ and of radius $k$. Note that $B_G(v, 1) = B_G(v)$.

$G'$ is a subgraph of $G$ if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. The subgraph of $G$ induced by a subset $V'$ of $V(G)$ is the subgraph of $G$ having $V'$ as vertex set and containing all edges of $G$ between vertices of $V'$.

A homomorphism between two graphs $G$ and $H$ is a mapping $\gamma : V(G) \to V(H)$ such that if $\{u, v\}$ is an edge of $G$ then $\{\gamma(u), \gamma(v)\}$ is an edge of $H$. Since we deal only with graphs without self-loops, this implies that $\gamma(u) \neq \gamma(v)$ if $\{u, v\}$ is an edge of $G$. Note also that $\gamma(N_G(u)) \subseteq N_H(\gamma(u))$. We say that $\gamma$ is an isomorphism if $\gamma$ is bijective and $\gamma^{-1}$ is also a homomorphism. By $G \simeq G'$ we mean that $G$ and $G'$ are isomorphic. A class of graphs will be any class of graphs in the set-theoretical sense containing all graphs isomorphic to some of its members. The class of all graphs will be denoted $\mathcal{G}$.

### 2.2. Labeled graphs

Throughout the paper we will consider only connected graphs where vertices are labeled with labels from a possibly infinite alphabet $L$. A graph labeled over $L$ will be denoted by $(G, \lambda)$, where $G$ is a graph and $\lambda : V(G) \to L$ is the function labeling vertices. The graph $G$ is called the underlying graph, and the mapping $\lambda$ is a labeling of $G$. The class of labeled graphs over some fixed alphabet $L$ will be denoted by $\mathcal{G}_L$.

Let $(G, \lambda)$ and $(G', \lambda')$ be two labeled 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 $\lambda$ is the restriction of the labeling $\lambda'$ to $V(G) \cup E(G)$.

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

An *occurrence* of $(G, \lambda)$ in $(G', \lambda')$ is an isomorphism $\varphi$ between $(G, \lambda)$ and a subgraph $(H, \eta)$ of $(G', \lambda')$.

### 2.3. Ambiguous graphs

A labeling is said *locally bijective* if vertices with the same label have isomorphic labeled neighbourhood. Formally,

**Definition 1** [5]. Let $L$ be a set of labels and let $(G, \lambda)$ be a labeled graph. The labeling $\lambda$ is called *locally bijective* if it verifies the following two conditions:

(1) For each $v \in V$ and for all $v', v'' \in B_G(v)$

$$\lambda(v') = \lambda(v'') \implies v' = v''.$$

(2) For all $v', v'' \in V$

$$\lambda(v') = \lambda(v'') \implies \lambda\big(N_G(v')\big) = \lambda\big(N_G(v'')\big).$$

A graph $G$ is said *ambiguous* if there exists a non-bijective labeling of $G$ which is locally bijective.

See [5] for examples of ambiguous graphs. Note that it is very unlikely for a graph to be ambiguous and that, in particular, trees, chordal graphs, graphs with a prime number of vertices are non-ambiguous.

## 3. Framework

We consider a network of processors with arbitrary topology. It is represented as a connected, undirected

graph where vertices denote processors, and edges denote direct communication links. Labels (or states) are attached to vertices. Labels are modified locally, that is, on a subgraph of the given graph, according to certain rules depending on the subgraph only (*local computations*). The relabeling is performed until no more transformation is possible, i.e., until a normal form is obtained. In this paper, local computations will be presented in the framework of graph relabeling systems that are very closed to the notion of *protocol* of [5].

### 3.1. Local computations

Local computations as considered here can be described in the following general framework [3]. Let $\mathcal{G}$ be the class of $L$-labeled graphs and let $\mathcal{R} \subseteq \mathcal{G} \times \mathcal{G}$ be a binary relation on $\mathcal{G}$. Then $\mathcal{R}$ will denote a graph rewriting relation. We assume that $\mathcal{R}$ is closed by isomorphism, i.e., whenever $(G,\lambda)\mathcal{R}(G',\lambda')$ if $(G_1,\lambda_1) \simeq (G,\lambda)$ then $(G_1,\lambda_1)\mathcal{R}(G'_1,\lambda'_1)$ for some labeled graph $(G'_1,\lambda'_1) \simeq (G',\lambda')$. In the remainder of this paper $\mathcal{R}^*$ stands for the transitive closure of $\mathcal{R}$.

**Definition 2.** Let $\mathcal{R} \subseteq \mathcal{G} \times \mathcal{G}$ be a graph rewriting relation.

(1) $\mathcal{R}$ is a *relabeling relation* if whenever two labeled graphs are in relation then the underlying graphs are equal, [1] i.e.:

$$(G,\lambda)\mathcal{R}(H,\lambda') \implies G = H.$$

(2) $\mathcal{R}$ is *local* if only labels of a ball may be changed by $\mathcal{R}$, i.e., $(G,\lambda)\mathcal{R}(G,\lambda')$ implies that there exists a vertex $v \in V(G)$ such that

$$\lambda(x) = \lambda'(x) \quad \text{for every } x \notin V\big(B_G(v)\big) \cup E\big(B_G(v)\big).$$

*Local computations*, where labels change locally using only local information, are described precisely in the next definition that states that a local relabeling relation $\mathcal{R}$ is *locally generated* if its restriction on centered balls determines its computation on any graph.

[1] We say equal, not only isomorphic: the next notions and definitions would be unnecessarily difficult to handle without equal underlying graphs.

**Definition 3.** Let $\mathcal{R}$ be a relabeling relation. Then $\mathcal{R}$ is *locally generated* if the following is satisfied: For every labeled graphs $(G,\lambda)$, $(G,\lambda')$, $(H,\eta)$, $(H,\eta')$ and every vertices $v \in V(G)$, $w \in V(H)$ 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) $\lambda(x) = \eta(\varphi(x))$ and $\lambda'(x) = \eta'(\varphi(x))$ for all $x \in V(B_G(v)) \cup E(B_G(v))$,
(2) $\lambda(x) = \lambda'(x)$, for all $x \notin V(B_G(v)) \cup E(B_G(v))$,
(3) $\eta(x) = \eta'(x)$, for all $x \notin V(B_H(w)) \cup E(B_H(w))$

imply that $(G,\lambda)\mathcal{R}(G,\lambda')$ if and only if $(H,\eta)\mathcal{R}(H,\eta')$.

In the following, we consider relation $\mathcal{R}$ that will always be locally generated by a recursive set of rules.

The labeled graph $(G,\lambda)$ is *$\mathcal{R}$-irreducible* if there is no $(G,\lambda')$ such that $(G,\lambda)\mathcal{R}(G,\lambda')$. Let $(G,\lambda) \in \mathcal{G}$, then $\mathrm{Irred}_{\mathcal{R}}((G,\lambda))$ denotes the set of $\mathcal{R}$-irreducible graphs (or irreducible if $\mathcal{R}$ is fixed) which can be obtained from $(G,\lambda)$ using $\mathcal{R}$. The relation $\mathcal{R}$ is noetherian if there is no infinite relabeling sequence $(G,\lambda_1)\mathcal{R}(G,\lambda_2)\mathcal{R}\ldots$.

### 3.2. Distributed computations of local computations

The notion of relabeling sequence defined above, obviously, corresponds to a notion of *sequential* computation. Let us also note that a locally generated relabeling relation also allows parallel rewritings, since non-overlapping balls may be relabeled independently. Thus we can define a distributed way of computing by saying that two consecutive relabeling steps concerning non-overlapping balls may be applied in any order. We say that such relabeling steps commute and they may be applied concurrently. More generally, any two relabeling sequences such that the latter one may be obtained from the former one by a succession of such commutations lead to the same resulting labeled graph. Hence, our notion of relabeling sequence may be regarded as a *serialization* [4] of some distributed computation. This model is clearly asynchronous: several relabeling 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 deal with sequential relabeling sequences but the reader should keep in mind that such sequences may be done in a distributed way.

### 3.3. Enumeration on graphs

The nodes uses a special *register* to describe their identity, it will be denoted ID.

**Definition 4.** A relabeling system $\mathcal{R}$ is said to self-stabilize to an *Enumeration Solution* on a graph $G$ if $\mathcal{R}$ is noetherian and for any initial labeling $\lambda_0$, the final labeling $(G, \lambda_f) \in \mathrm{Irred}((G, \lambda_0))$ of any relabeling chain is locally bijective and is of the form $\lambda_f(v) = (\mathrm{ID}(v), c(v))$ where
- ID is a bijection from $V(G)$ to $\{1, \ldots, |V(G)|\}$.
- $c$ is any mapping to $L$.

**Theorem 5** [5]. *If $G$ is ambiguous then there is no relabeling system stabilizing to an enumeration solution on $G$.*

**Proof.** Theorem 2 of [5] states it is true even if we require only uniform initial labelings. □

In the following section we present a relabeling system that, on *non-ambiguous graphs*, self-stabilizes to an enumeration of the vertices.

## 4. Self-stabilizing enumeration algorithm

This result will be addressed in the following way: first we present our modified version of Mazurkiewicz' Enumeration Algorithm. We will prove an important characterization of the final labelings obtained with this algorithm. Then we describe a procedure for enumerating nodes that uses this final labeling. Finally we give some bounds on the time of stabilization.

### 4.1. Modifying Mazurkiewicz' Enumeration Algorithm

An enumeration algorithm on a network $G$ is a distributed algorithm such that the result of any computation is a labeling 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 [5], the computation model is equivalent even thought it is described in the different formalism of protocols. Our self-stabilizing version will be denoted $\mathcal{M}$.

**Description.** We give first a general description of the algorithm $\mathcal{M}$. Every vertex attempts to get its own name, which shall be an integer between 1 and $|V(G)|$. A vertex chooses a name and broadcasts it together with its neighborhood all over the network. If a vertex $u$ discovers the existence of another vertex $v$ with the same name, then it compares its *local view*, i.e., the labeled ball of center $u$, with the local view of its rival $v$. If the local view of $v$ is "stronger", then $u$ chooses another name. $u$ also chooses another name if its appears twice in the view of some other vertex as a result of a corrupted initial state. Each new name is broadcast again over the network. At the end of the computation it is not guaranteed that every node has a unique name, unless the graph is non-ambiguous. However, all nodes with the same name will have the same local view, i.e., isomorphic labeled neighborhoods. These names cannot be straight used as identities because they can be arbitrary large, but it is possible to compute the rank of the name in all the existing names and set it as its ID.

The crucial property of the algorithm is based on a total order on local views such that the "strength" of the local view of any vertex cannot decrease during the computation. To describe the local view we use the following notation: if $v$ has degree $d$ and its neighbors have names $n_1, \ldots, n_d$, with $n_1 \geqslant \cdots \geqslant n_d$, then $N(v)$, the local view, is the $d$-tuple $(n_1, \ldots, n_d)$. Let $\mathcal{L}$ be the set of such ordered tuples.

The alphabetic order defines a total order, $\preccurlyeq$, on $\mathcal{L}$.

Vertices $v$ are labeled by triples of the form $(n, \overline{N}, M)$ representing during the computation:
- $n(v) \in \mathbb{N}$ is the name of the vertex $v$,
- $\overline{N}(v) \in \mathcal{L}$ is the latest view of $v$,
- $M(v) \subset \mathbb{N} \times \mathcal{L}$ is the mailbox of $v$ and contains all information received at this step of the computation.

We need other notations. We want to count the number of times a given name appear in a local view. For a local view $N$, and $n \in \mathbb{N}$, we define $\delta_N(n)$ to be the cardinality of $n$ in the tuple $N$.

For a given view $N$, we denote by $\mathrm{sub}(N, n, n')$ the copy of $N$ where any occurrence of $n$ is replaced by $n'$.

We recall that in the self-stabilizing context, the initial labels of all vertices are arbitrary elements in the set of labels $L$.

The rules are described below for a given centered ball $B = B(v_0)$ with center $v_0$. The vertices $v$ of

$B$ have labels $(n(v), M(v))$. $N(v_0)$ is the local view of $v_0$, hence the ordered tuple of the names of its neighbors. The labels obtained after applying a rule are $(n'(v), M'(v))$. To make the rules easier to be read, we omit labels that are left unchanged.

**Diffusion rule:**

*Precondition*:

$$\exists v, M(v) \neq M(v_0) \tag{1a}$$
$$\vee$$
$$\big(n(v_0), N(v_0)\big) \notin M(v_0) \tag{1b}$$
$$\vee$$
$$\overline{N}(v_0) \neq N(v_0) \tag{1c}$$

*Relabeling*:

– For all $v$,
$$M'(v) := \bigcup_{w \in B} M(w) \cup \big(n(v_0), N(v_0)\big),$$
– $\overline{N}'(v_0) := N(v_0)$.

**Renaming rule:**

*Precondition*:

$$\big(n(v_0), N(v_0)\big) \in M(v_0) \tag{2a}$$
$$\wedge$$
$$\overline{N}(v_0) = N(v_0) \tag{2b}$$
$$\wedge$$
$$\forall v,\ M(v) = M(v_0) \tag{2c}$$
$$\wedge \ \{$$
$$n(v_0) = 0 \tag{2d}$$
$$\vee$$
$$n(v_0) > 0 \text{ and } \exists N_1\ (n(v_0), N_1) \in M(v_0) \text{ and } N(v_0) \prec N_1 \tag{2e}$$
$$\vee$$
$$n(v_0) > 0 \text{ and } \exists (n_1, N_1) \in M(v_0) \text{ such that } \delta_{N_1}\big(n(v_0)\big) \geqslant 2 \tag{2f}$$
$$\}$$

*Relabeling*:

– $n'(v_0) := 1 + \max\{n \in \mathbb{N} \mid (n, N) \in M(v_0) \text{ for some } N \in \mathcal{L}\}$,
– $\overline{N}'(v) := \text{sub}(\overline{N}(v), n(v_0), n'(v_0))$,
– The mailbox contents changes to
$$M'(v) := M(v_0) \cup \big\{(n'(v_0), N(v_0))\big\} \bigcup_{w \in B} \big(n(w), \overline{N}'(w)\big).$$

Since the IDs are not used by $\mathcal{M}$, their computation is described later, in Section 4.3.

### 4.2. Properties of $\mathcal{M}$

The proof will follow the scheme of the one used in [5]. Let $G$ be a graph, let us denote by $(n_i(v), M_i(v))$ the label of a vertex $v$ of $G$ and by $N_i(v)$ the local view of $v$ after the $i$th step of the computation of $\mathcal{M}$. See the remark on serialization Section 3.2.

We first prove some lemmas. We suppose the step $i$ is fixed.

We say that a name $m$ is known by $v$, if $(m, N) \in M(v)$ for some $N$. Let us note $r$ the greatest of all names known to the graph nodes at step 0. For showing termination, we claim that whenever a name $m$ is known and $m > r$, there is a vertex having this name.

**Lemma 6.** *$\forall v \in V(G)\ \forall m > r$ such that $\exists N(m, N) \in M_i(v_0)$, $\exists w \in V(G)$ such that $n_i(w) = m$.*

**Proof.** Assume that the name $m > r$ is known by $v_0$ and let $U = \{u \in V(G) \mid \exists j < i,\ n_j(u) = m\}$, then $U$ is not empty because, as $m > r$, there has to be a vertex getting this name by *Relabeling rule*. Let $w$ be the vertex with the strongest neighbourhood in $U$, then the *Renaming rule* cannot be applied to $w$, hence $n_i(w) = m$. □

**Lemma 7.** *$\forall v \in V(G)\ \forall m > r$, $\delta_{N(v)}(m) \leqslant 1$.*

**Proof.** Suppose the assertion is false, then there exists $m$ and $v$ with two neighbors named $m$. As there were no $m$ in the initial labeling, $m$ had to be set by former applications of *Relabeling rule*. Consider the vertex $w_2$ having been last named $m$. $w_2$ has $v$ as a neighbor and the first application of *Relabeling rule* should have made $m$ known by $v$, thus by $w_2$ before it can apply any *Relabeling rule*. Yields a contradiction. □

Next, we claim that whenever a name greater than $r$ is known, all positive names greater than $r$ and smaller than that name are assigned to some vertex.

**Lemma 8.** *$\forall i,\ \forall v \in V(G)$ such that $n_i(v) \neq r\ \forall m$, $r \leqslant m \leqslant n_i(v)$, $\exists w \in V(G)$ 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 $i \geqslant 0$. If *Diffusion rule* is used, the assertion

is true for $i+1$. If we use *Renaming rule* applied to $v_0$ then we just have to verify it for $v_0$, and more precisely for all names $m \in [n_i(v_0), \ldots, n_{i+1}(v_0)] \cap [r, +\infty[$. The property for $n_{i+1}(v_0)$ is clear and $n_i(v_0)$ will be treated later.

If the interval $[n_i(v_0)+1, \ldots, n_{i+1}(v_0)-1] \cap [r, +\infty[$ is empty then the condition is obviously satisfied. Otherwise by definition of the *Renaming rule* and the previous claim there exists $w \neq v_0$ such that $n_i(w) = n_{i+1}(w) = n_{i+1}(v_0) - 1$. For $m \in [n_i(v_0)+1, \ldots, n_{i+1}(v_0)-1]$, the assertion is then true by induction hypothesis on $w$.

To prove the assertion for $n_i(v_0)$, first we remark that if $n_i(v_0) > r$, by Lemma 7, we are in subcondition (2e) and the assertion is a direct consequence of Lemma 6's proof: the vertex with the strongest neighbourhood in $U$ has $n_i(v_0)$ as a name at step $i+1$. □

The labeling function obtained at the end of a run $\mathcal{E}$ of Mazurkiewicz' algorithm is noted $\Pi_{\mathcal{E}}$. If $v$ is a vertex of $G$, the couple $\Pi_{\mathcal{E}}(v)$ associated with $v$ is denoted $(n_{\mathcal{E}}(v), M_{\mathcal{E}}(v))$. We also note the final local view of $v$ by $N_{\mathcal{E}}(v)$. For a given mailbox $M$ and a given $n \in \mathbb{N}$, we note $N_M(n)$ the local view such that for all $N$, $(n, N) \in M \Rightarrow N \preccurlyeq N_M(n)$. Except for the first corrupted stages, $N_{M(v)}(n)$ is actually the "strongest local view" of $n$.

**Theorem 9.** *A run of Mazurkiewicz' Enumeration Algorithm on a connected graph $G$ with any initial labeling finishes and computes a final labeling $\Pi_{\mathcal{E}}$ verifying the following conditions for all vertices $v, v'$ of $G$:*
(1) $M_{\mathcal{E}}(v) = M_{\mathcal{E}}(v')$.
(2) $N_{M_{\mathcal{E}}(v')}(n_{\mathcal{E}}(v)) = \overline{N}(v) = N_{\mathcal{E}}(v)$.
(3) *$n_{\mathcal{E}}$ induces a locally bijective labeling of $G$.*

**Proof.** Let $t = r + n$. Let $I = \{0, \ldots, t\}$ ordered by the natural order on integers, $\mathcal{L}_I$ be the set of the $I$-labeled balls of $G$, ordered by $\preccurlyeq$. $\mathcal{P}(I \times \mathcal{L}_I)$ is ordered by inclusion. Let $X = I \times \mathcal{P}(I \times \mathcal{L}_I)$. The product order $\precsim$ defines a partial order on $X^{V(G)}$. We note $x^i = ((n_i(v), M_i(v)))_{v \in V(G)}$, we will show that $(x^i)_i$ is a strictly increasing sequence of $X$.

First we claim that, for each $v, i$,
– $n_i(v) \leqslant n_{i+1}(v)$,
– $M_i(v) \subseteq M_{i+1}(v)$.

It is obviously true for the vertices that are not concerned by the rule applied at step $i$, and for the other vertices, we note that *Renaming rule* only increments $n_i(v_0)$, that *Diffusion rule* only adds elements to the mailboxes.

Moreover, one of the inequalities is strict for at least one vertex, namely the one for which the previous relabeling rule was applied. Then $x^i \precsim x^{i+1}$ for all $i$.

As, by Lemma 8, there is no more that $t$ different assigned $n$-labels at a given step, $x^i \in X$, for all $i$. We can now conclude for $X$ is a finite ordered space. The sequence $x^i$ is finite and the algorithm terminates.

The properties of the final labeling are easily derived from the fact that no more rule could be applied.
(1) Otherwise, *Diffusion rule* could be applied.
(2) Remark that for all $v$, $(n_{\mathcal{E}}(v), N_{\mathcal{E}}(v)) \in M_{\mathcal{E}}(v) = M_{\mathcal{E}}(v')$, otherwise precondition (1b) of *Diffusion rule* would hold. Then, as the precondition (2e) does not hold, we get the property.
(3) Otherwise *Renaming rule* could be applied to vertices having the same name and non-isomorphic local views. Remember that $\preccurlyeq$ is a *total* order. □

### 4.3. Enumeration

Now, each vertex shall compute locally the set of final names from the final mailbox $M_{\mathcal{E}}$. We note $G_{\mathcal{E}}$ the graph defined by

$$V_{\mathcal{E}} = \{n_{\mathcal{E}}(v) \mid v \in V(G)\},$$
$$E_{\mathcal{E}} = \{(n_{\mathcal{E}}(v_1), n_{\mathcal{E}}(v_2)) \mid (v_1, v_2) \in E(G)\}.$$

For a mailbox $M$ and and integer $n$, we define the set $V^M(n)$ by induction.

$$V_0^M = \{n\},$$
$$V_{i+1}^M = V_i^M \cup \{t \mid \exists s \in V_i^M,\ \delta_{N_M(s)}(t) = 1\}.$$

If $i_0$ is such that $V_{i_0}^M = V_{i_0+1}^M$ then we define $V^M(n) = V_{i_0}^M$.

Finally, we have,

**Lemma 10.** *For all $u$, $V^{M_{\mathcal{E}}}(n_{\mathcal{E}}(u)) = V_{\mathcal{E}}$.*

**Proof.** Let $n = n_{\mathcal{E}}(v)$ for some $v$. By induction, we prove that $V_i^{M_{\mathcal{E}}} = B_{G_{\mathcal{E}}}(n, i)$ for all $i$. It is obviously true for $i = 0$. Suppose it is true for $i$. Then by definition and by Theorem 9(2), $V_{i+1}^M \subset B_{G_{\mathcal{E}}}(n, i)$. Let $m \in B_{G_{\mathcal{E}}}(n, i+1) \setminus B_{G_{\mathcal{E}}}(n, i)$, $m$ has a neighbor

in $B_{G_{\mathcal{E}}}(n,i) = V_i^M$ by induction hypothesis. Hence $m \in V_{i+1}^M$ and finally $V^{M_{\mathcal{E}}}(n_{\mathcal{E}}(u)) = V_{\mathcal{E}}$. □

And at any time, each vertex $u$ set its ID by computing its *enumeration index*, i.e., the rank of $n(u)$ in the ordered set $V^{M(u)}(n(u))$. Lemma 10 means that the ID registers finally get a correct attribution. Thus, as presented in [5], in the case of non-ambiguous graphs, after a run of Mazurkiewicz' algorithm, ID induces a one-to-one correspondence between the set of vertices of $G$ and the set of integers $\{1,\ldots,|V(G)|\}$.

### 4.4. Complexity

The detection of termination is not relevant in the self-stabilization context because a corrupted initial state could made some node to believe they are on a final state, the important problem is then to compute the time of stabilization. In this section, we prove a bound of $O(t|v(G)|^2)$ steps, where $t$ is the sum of the number of vertices and of the highest name initially known.

**Proposition 11.** *An execution of $\mathcal{M}$ on a graph $G$ has at most $t \times |V(G)|^2$ relabeling steps.*

**Proof.** As there are at most $t$ $n$-labels for every vertex, the relabeling rule is applied at most $t \times |V(G)|$ times. And for any relabeling, there are at most $|V(G)|$ applications of Diffusion rule case (1a), one for each other node. Furthermore, case (1b) or (1c) appears at most once for each vertex, because once a vertex $v$ apply the diffusion rule, $(n(v), N(v)) \in M(v)$ and $\overline{N}(v)$ remains correct (simple induction[2]). Hence, every vertex is involved in a Diffusion at most the number of applications of Relabeling rules. Hence the bound yields. □

Thus, with correct initial labels, the algorithm stabilizes in at most $|V(G)|^3$ steps.

## 5. Conclusion

The extension of Mazurkiewicz' Algorithm presented here has interesting self-stabilizing properties. Furthermore, the quite low time of stabilization shows that on non-ambiguous graphs, it should be possible to design robust algorithms on the basis on the self-stabilizing identities provided by the algorithm presented in this paper.

## Acknowledgements

The author wish to thanks the anonymous referees for fruitful comments and observations.

## References

[1] E.W. Dijkstra, Self-stabilizing systems in spite of distributed control, Comm. ACM 17 (11) (1974) 643–644.
[2] E. Godard, Y. Métivier, A characterization of classes of graphs recognizable by local computations with initial knowledge (ext. abstract), in: SIROCCO 8, 8th International Colloqium on Structural Information and Communication Complexity, Proceedings in Informatics, Vol. 11, Carleton Scientific, Ottawa, ON, 2001, pp. 179–194.
[3] I. Litovsky, Y. Métivier, W. Zielonka, On the recognition of families of graphs with local computations, Inform. and Comput. 118 (1) (1995) 110–119.
[4] A. Mazurkiewicz, Trace theory, in: W. Brauer et al. (Eds.), Petri Nets, Applications and Relationship to Other Models of Concurrency, Lecture Notes in Comput. Sci., Vol. 255, Springer, Berlin, 1987, pp. 279–324.
[5] A. Mazurkiewicz, Distributed enumeration, Inform. Process. Lett. 61 (1997) 233–239.

[2] Note that $\overline{N}$ is maintained only for this complexity purpose.