A Local Self-Stabilizing Enumeration Algorithm

Brahim Hamid and Mohamed Mosbah

LaBRI-
Université Bordeaux-1
351, cours de la libération
F-33405 Talence Cedex, France
Email: {hamid,mosbah}@labri.fr

Abstract. We present a novel self-stabilizing version of Mazurkiewicz enumeration algorithm [1]. The initial version is based on local rules to enumerate nodes on an anonymous network. [2] presented the first self-stabilizing version of this algorithm which tolerates transient failures with an extension of messages complexity. Our version is based on local detection and correction of transient failures. Therefore, it ensures the fault-tolerance property without adding messages or reduces the messages' number of other version. In addition, we have developed an interface based on the Visidia platform to simulate faults through a graphical user interface. The implementation of the presented algorithm in this platform shows its dynamic execution and validates its correction.

1 Introduction

Distributed computing systems are becoming larger and larger, heterogeneous and complex. Since the applications running on these systems require the cooperation of many components, they are prone to failures and errors of many different types, leading to inconsistent executions. Hence, a desirable feature of a computation in a distributed system is fault-tolerance. A particularly suitable approach to deal with such a feature is to design self-stabilizing algorithms [3, 4].

The concept of self-stabilization [5] is introduced to design a system which tolerates transient failures. Informally, self-stabilizing algorithms ensure that after any failure, the system will automatically recover to reach a correct configuration in a finite time. In general, self-stabilizing algorithms are constructed in such a way that a given process will continue to function correctly in spite of intermittent faults.

An anonymous network is a network where all nodes execute the same algorithm without a unique identity for each node. In general, the task solved by an enumeration algorithm is the affectation of a different *name* to each node of an anonymous network. So, such algorithm may be used as a preprocessing task of many algorithms based on the identities. As well-known, many problems have no solutions in anonymous networks. The motivations of this work are in the first hand to design an enumeration protocol in anonymous networks using

only local computations [1]. On the other hand, we show the adaptation of our developed framework [6] to enumeration algorithm in the presence of transient failures.

We are interested on the study of the Mazurkiewicz enumeration algorithm [1] based on local computations. Mazurkiewicz's algorithm is a distributed algorithm to enumerate nodes in an anonymous minimal-covering graph when its size is known. A distributed enumeration algorithm on a graph G is a distributed algorithm such that the result of any computation is a labeling of the nodes that is a bijection from $V(G)$ to $1, 2, ..., |V(G)|$. [2] proposed a version of self-stabilizing enumeration algorithm with a final stage in which each node computes locally the set of final names from the final mailbox. Before this stage the node can choose a name which is greater than the size of the graph.

Many self-stabilizing algorithms have been already designed [7, 8]. However, most of these works propose global solutions which require to involve the entire system. As networks grow fast, detecting and correcting errors globally is no longer feasible. The solutions that deal locally with detection and correction are rather essential because they are scalable and can be deployed even for large and evolving networks. Moreover, it is useful to have the correct (non faulty) parts of the network operating normally while recovering locally the faulty components. Few general approaches providing local solutions to self-stabilization have been proposed in [9, 10, **?**].

In [6], we consider the problem of designing algorithms encoded by local computations in a distributed system with transient failures. The developed formal framework allows to design and prove fault-tolerant distributed algorithms. We introduce correction rules which can be applied by faulty nodes or by their neighbors in order to self repair the incorrect states. Such rules have higher priorities than the main rules of the algorithm which ensure that the failures are repaired before continuing the computation. Of course, we deal only with predefined faulty local configurations. A tool called *Visidia* [11], validating the local computations model has been implemented. The distributed system of Visidia is based on asynchronous message passing model. However, it has been assumed that components of such a system do not fail. In this work, we show a simulation of fault-tolerant enumeration algorithm using this platform.

The paper is organized as follows. Models of distributed systems and graph relabeling systems and our framework are presented in Section 2. We give in Section 3 our solution to encode self-stabilizing enumeration algorithm. Section 4 gives its analysis and Section 5 shows an implementation of our protocol on Visidia platform. Finally Section 6 concludes the paper.

2 Preliminaries

2.1 The Model of Distributed System

A distributed system is modeled by a graph $G = (V, E)$, where $V(G)$ is the set of nodes and $E(G)$ is the set of edges. Nodes represent processes and edges represent

bidirectional communication links. Two nodes are connected by an edge if the corresponding processes have a direct communication link. We denote by $N_G(v)$ the set of neighbors of v in the graph G, that is, $\forall\ u \in N_G(v)$, $(v,u) \in E(G)$. A ball center on u with radius 1 is the set $B(u) = \{u\} \cup N_G(u)$. The cardinality of set $V(G)$ (which is also the size of the corresponding network) is denoted by $\mid V(G) \mid$, and we assume that $\mid V(G) \mid = \mathcal{N}$. For any set A, we write 2^A (resp. A^n) to denote the set of all finite subsets of A (resp. the set of all n-tuples, for $n \in \mathcal{N}$ of element of A), where $\mathcal{N}$ is the set of all positives integers. In the considered networks, processes communicate and synchronize by sending and receiving messages through the links. There is no assumption about the relative speed of processes or message transfer delay, the networks are *asynchronous*. The topology is unknown and each node communicates only with its neighbors. The links are reliable and the process can fail and recover in a finite time. The failures that are tolerated in such a system are the transient failures of processes.

The set $N_G(v)$ (resp.$N_h(w)$) is composed of all the neighbors of v in the graph G (resp. the neighbors of w in the graph H). We say that a graph G is a *covering* of a graph H if there exists a surjective homomorphism φ from G onto H such that for every node v of $V(G)$ the restriction of φ to $N_G(v)$ is a bijection onto $N_H(\varphi(v))$. In particular, $(\varphi(v), \varphi(u)) \in E(H)$ implies $(v,u) \in E(G)$. The covering is *proper* if G and H are not isomorphic. It is called connected if G (and thus also H) is connected. A graph G is called *minimal-covering* if every covering from G to some H is a bijection.

The stabilizing algorithms are optimistic, they guarantee a return to a correct behavior within a finite time after all faulty behaviors cease. Self-stabilizing algorithms protect against transient failures, since they can automatically repair any fault in the system. The term *fault* refers to failure.

2.2 Graph Relabeling Systems (GRS) to Encode Distributed Algorithms

Local computations, and particularly graph relabeling systems [12] are a powerful model which provides general tools to encode distributed algorithms, to prove their correctness and to understand their power. In such a model we consider a network of processes with arbitrary topology represented as a connected, undirected graph where nodes denote processes, and edges denote communication links. Every time, each node and each edge is in some particular state and this state will be encoded by a node label or an edge label. According to its own state and to the states of its neighbors, each node may decide to realize an elementary *computation step*. After this step, the states of this node, of its neighbors and of the corresponding edges may have changed according to some specific *computation rules*. Let us recall that graph relabeling systems satisfy the following requirements:

(C1) they do not change the underlying graph but only the labeling of its components (edges and/or nodes), the final labeling being the result,

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

(C3) they are locally generated, that is, the applicability condition of the relabeling only depends on the local context of the relabeled subgraph.

Let L be an alphabet and let G be a graph. We denote by (G, λ) a graph G with a relabeling function $\lambda : V(G) \cup E(G) \to L$. A labeling is said to be locally bijective if nodes with the same label have isomorphic labeled neighbors. Then, the graph G is said to be ambiguous if there exists a non bijective labeling of G which is locally bijective. For more examples the reader is referred to [1].

A graph relabeling system is a triple $\Re = (L, I, P)$ where L is a set of labels, I is a subset of L called the set of initial labels and P a finite set of relabeling rules. Consider an arbitrary system $\Re = (L, I, P)$ and a labeling function λ. A relabeling step will be denoted by $(G, \lambda) \underset{\mathcal{R}}{\longrightarrow} (G, \lambda')$. The notion of computation then corresponds to the notion of relabeling sequence. A relabeling sequence with any steps will be denoted by $(G, \lambda) \xrightarrow[\mathcal{R}]{*} (G, \lambda')$. A relabeling sequence with k steps will be denoted by $(G, \lambda) \xrightarrow[\mathcal{R}]{k} (G, \lambda')$.

The program is encoded by a graph relabeling system $\Re = (L, I, P)$. The labels of each process represent the values of its variables. Each rule in set P is of the following form:

$$R1 : \mathbf{RuleN}\{Precondition\}\{Relabeling\}$$

$R1$ denotes the number of the rule and **RuleN** is the name of the rule. The *Precondition* part of a rule in the program of v_0 is a boolean expression (predicate) involving the labels of v_0 and the labels of its neighbors. The *Relabeling* part of a rule of v_0 updates one or more labels of v_0 and its neighbors. A rule can be executed only if its precondition is *true*. The rules are atomically executed, meaning that, the evaluation of a precondition and the execution of its corresponding relabeling, if the precondition is *true*, are done in one atomic step.

2.3 Self-stabilizing Graph Relabeling Systems

An algorithm is called self-stabilizing if it eventually starts to behave correct according to its specifications regardless of its initial configuration [5]. A *local configuration* of a process is composed by its state, the states of its neighbors and the states of its communication links. In this work we use the notion of local illegitimate configurations encoded by local computations [6]. For a labeled graph (G, λ), we say that a local configuration $f = (B_f, \lambda_f)$ is illegitimate for (G, λ), if there is no subgraph in (G, λ) which is isomorphic to f. In other words, there is no ball (neither sub-ball) of radius 1 in G which has the same labeling as f. Such labels are not used when the system runs in a correct manner.

Transient failures cause processes to change their states yielding illegitimate local configurations and therefore an illegitimate global configuration. A self-stabilizing system will be able to destroy such a fault by eventually stabilizing into a correct global configuration without restarting the system. A local stabilizing graph relabeling system is a triple $\Re = (L, \mathcal{P}, \mathcal{F})$ where L is a set of labels,

$\mathcal{P}$ a finite set of relabeling rules and $\mathcal{F}$ is a set of illegitimate local configurations [6]. Let $\mathcal{G}_{\mathcal{L}}$ be the set of labeled graphs (G, λ) and $h : \mathcal{G}_{\mathcal{L}} \longrightarrow \mathbb{N}$ be an application associating to each labeled graph (G, λ), the number of its illegitimate configurations at this stage of the computation. Therefore, we denote by $h(G, \lambda, \mathcal{F})$ the current number of illegitimate configurations of the labeled graph (G, λ). A local stabilizing graph relabeling system must satisfy the two following properties:

- *Closure* : $\forall\ (G, \lambda) \in \mathcal{G}_{\mathcal{L}}$ such that $h(G, \lambda, \mathcal{F}) = 0$,
  $\forall\ (G, \lambda')\ /(G, \lambda) \xrightarrow[\Re]{*} (G, \lambda') : h(G, \lambda', \mathcal{F}) = 0$.
- *Convergence* : $\forall\ (G, \lambda) \in \mathcal{G}_{\mathcal{L}}, \exists$ an integer k, such that $(G, \lambda) \xrightarrow[\Re]{k} (G, \lambda')$ and $h(G, \lambda', \mathcal{F}) = 0$.

As for self-stabilizing algorithms, the closure property stipulates the correctness of the relabeling system. A computation beginning in a correct state remains correct until the terminal state. The convergence however provides the ability of the relabeling system to recover automatically within a finite time (finite sequence of relabeling steps).

In [6] we state the following result: if $\Re = (L, I, P, \mathcal{F})$ is a graph relabeling system with illegitimate configurations $\mathcal{F}$, then it can be transformed into an equivalent local stabilizing graph relabeling system $\Re_s = (L, P_s, \mathcal{F})$. The set P_s is composed of set P and some correction rules to detect and eliminate each illegitimate configuration of $\mathcal{F}$. The correction rules have higher priority than the rules in P.

3 The Local Stabilizing Enumeration Algorithm

3.1 The Mazurkiewicz's Enumeration Algorithm

An enumeration algorithm on a graph G is a distributed algorithm such that the result of any computation is a labeling of the nodes that is a bijection from $V(G)$ to $1, 2, ..., |V(G)|$. First, we give a description of the initial enumeration algorithm [1]. Every node attempts to get its own name, which shall be an integer between 1 and $|V(G)|$. A node chooses a name and broadcasts it with its neighbor-hood (i.e. the list of the name of its neighbors) all over the network. If a node u discovers the existence of another node 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 v is "Stronger", then u chooses another name. Each new name is broadcast with the local view 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.

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 node cannot decrease during the computation. To describe this local view we use the following notation: if

v has degree d and its neighbors have names $n_1, n_2, ...n_d$ with $n_1 \geq .... \geq n_d$, then $LV(v)$, the local view, is the *d-uplet*$(n_1, n_2, ...n_d)$. Let $\mathcal{LV}$ be the set of such ordered tuples. The alphabetic order defines a total order $\preceq$ on $\mathcal{LV}$. The nodes v are labeled by triples of the form (n, LV, GV) representing during the computation :

- $n(v) \in \mathbb{N}$ is the name of the node v,
- $LV(v) \in \mathcal{LV}$ is the latest view of v,
- $GV(v) \subset \mathbb{N} \times \mathcal{LV}$ is the mailbox of v and contains all the information received at this step of the computation. We call this set the global view of the v.

We define the list $sub(LV, n, n')$: the copy of LV where any occurrence of n is replaced by n' if n exists or adds n to LV otherwise. Let $LV \in \mathcal{LV}$ and $(n, n') \in \mathbb{N}^2$, if $n < n'$ then $LV \prec sub(LV, n, n')$. The initial labels of each node are $(0, \phi, \phi)$. Each node v has labels: $(n(v), LV(v), GV(v))$ and the labels obtained after applying a rule are $(n'(v), LV'(v), GV'(v))$. Let v_0 a node which is center of ball $B(v_0)$ and let $\{v_1, v_1 \cdots, v_d\}$ the set of its neighbors. Let $(n(v_i), LV(v_i), GV(v_i))$ the triple associated to the node v_i with $0 \leq i \leq d$. We call the ball of the center v and we write $B(v)$ the set composed of v and its neighbors. Now we present Mazurkiewicz's enumeration algorithm:

R1 : **Transmitting rule**

Precondition :

- $\exists\ v_i \in B(v_0), GV(v_i) \neq GV(v_0)$

Relabeling :

- $\forall\ v_i \in B(v_0), GV'(v_i) := \bigcup_{v_j \in B(v_0)} GV(v_j)$

R2 : **Renaming rule**

Precondition :

- $\forall\ v_i \in B(v_0), GV(v_i) = GV(v_0)$
- $n(v_0) = 0$ or
  $n(v_0) > 0$ and $(\exists\ LV_1 | (n(v_0), LV_1) \in GV(v_0)$ and $LV(v_0) \prec LV_1)$

Relabeling :

- $n'(v_0) := 1 + max\{n \mid (n, LV) \in GV(v_0)\}$
- $\forall\ v_i \in B(v_0) \backslash \{v_0\}, LV'(v_i) := sub(LV(v_i), n(v_0), n'(v_0))$
- $\forall\ v_i \in B(v_0), GV'(v_i) := GV(v_i) \bigcup_{v_j \in B(v_0)} \{(n'(v_j), LV'(v_j))\}$

3.2 A Local Stabilizing Enumeration Algorithm

We present in the sequel a new enumeration algorithm encoded by local stabilizing relabeling systems. This protocol is optimal compared to that of [2]. In a correct behavior, when a name of v_0 is already chosen by another node v_i, v_0 (resp. v_i) will receive this information and change its name if the local view of v_0 (resp. v_i) contains the older modifications. In a corrupted behavior, when a name of v_0 is corrupted, v_0 detects this corruption, change its name to -1

and initialize its states, then one of its neighbors v_i detects this change, corrects some of the state of v_0. After that, v_0 chooses another number to rename itself.

Function $\delta_L(n)$ gives the number of occurrences of a name n in the list LV. We start by defining some illegitimate configurations to construct a set $\mathcal{F}$, then we improve the system by adding the correction rules to detect and to eliminate these configurations. The node v_0 is said to be corrupted or in the illegitimate configuration, if one of its components is changed using extra relabeling. This relabeling does not correspond to those of the previous rules. We can define the following predicates to denote theses behaviors.

1. Corruption of the name: $n(v_0) \neq 1+max\{n \mid (n, LV) \in GV(v_0)\}$ or $n(v_0) >\mid V(G) \mid$.
2. Corruption of the local view:$\exists\ n_1 \in LV(v_0) \mid \neg\exists\ v_i \in B(v_0)\backslash\{v_0\} : n(v_i) = n_1$ or $n_1 >\mid V(G) \mid$.
3. Corruption of the global view:$\exists\ (n_1, LV_1) \in GV(v_0),\ \delta_{LV_1}(n(v_0)) \geq 2$ or $n_1 >\mid V(G) \mid$.

The function $choose\_unused(GV)$ chooses one unused name in the set of global view $GV \subset \mathbb{N} \times \mathcal{LV}$. We use the list $subset(GV, (n, LV), (n', LV'))$ to denote the copy of GV where any occurrence of (n, LV) is replaced by (n', LV') if $(n, LV) \in GV$ or adds (n', LV') to GV. For the present system, we deal with the following set $\mathcal{F} = \{f_1, f_2, f_3\}$ encoding the previous behaviors' predicates. Therefore, the correction rules are:

RC1 : **Corruption of the name**

Precondition :

- $n(v_0) \neq 1 + max\{n \mid (n, LV) \in GV(v_0)\}$ or $n(v_0) >\mid V(G) \mid$

Relabeling :

- $(n'(v_0), LV'(v_0), GV'(v_0)) := (-1, \phi, \phi)$
- $\forall\ v_i \in B(v_0)\backslash\{v_0\}, LV'(v_i) := LV(v_i)\backslash\{n(v_0)\}$
- $\forall\ v_i \in B(v_0), GV'(v_i) := GV(v_i)\backslash\{(n(v_0), LV(v_0))\}$

RC2 : **Choose of the name**

Precondition :

- $n(v_0) = -1$

Relabeling :

- $n'(v_0) := choose\_unused(GV(v_0))$
- $\forall\ v_i \in B(v_0)\backslash\{v_0\}, LV'(v_i) := sub(LV(v_i), n(v_0), n'(v_0))$
- $\forall\ v_i \in B(v_0), GV'(v_i) := GV(v_i) \bigcup_{v_j \in B(v_0)} \{(n'(v_j), LV'(v_j))\}$

RC3 : **Corruption of the local view**

Precondition :

- $\exists\ n_1 \in LV(v_0) \mid \neg\exists\ v_i \in B(v_0)\backslash\{v_0\} : n(v_i) = n_1$ or $n_1 >\mid V(G) \mid$

Relabeling :

- $LV'(v_0) := LV(V_0)\backslash\{n_1\}$
- $GV'(v_i) := subset(GV(v_i), (n(v_0), LV(v_0)), (n(v_0), LV'(v_0)))$

RC4 : **Corruption of the global view**

Precondition :

- $\exists\ (n_1, LV_1) \in GV(v_0)$, $\delta_{LV_1}(n(v_0)) \geq 2$ or $n_1 >\mid V(G) \mid$

Relabeling :

- $GV'(v_0) := GV(v_0) \backslash \{(n_1, LV_1)\}$

4 Analysis

4.1 Properties

We define the relabeling system $\Re_s = (L, P_s, \mathcal{F})$, where $L = \{\{\mathcal{N} \cup \{0\}\} \times 2^{\mathcal{N}} \times 2^{\mathcal{N} \times 2^{\mathcal{N}}}\}$ and $P_s = \{R1, R2, Rc1, Rc2, Rc3, Rc4\}$ such that $Rc_j > R_i$. We now state the main results.

Lemma 1. *The system $\Re_s = (L, P_s, \mathcal{F})$ satisfies the closure property.*

Proof. We prove this Lemma by induction on the size of relabeling sequences. Let (G, λ) any labeled graph where $h(G, \lambda, \mathcal{F}) = 0$. Let (G, λ_k) a labeled graph obtained from (G, λ) by applying k rules only from the set $P = \{R1, R2\}$. From the definition of correction rules, they are applied when some illegitimate configurations are introduced. When $k = 0$ the Lemma is *true*. We suppose that the lemma remains *true* after applying k rules. Now we show that the lemma remains true after the application of $k+1$ rules. From the induction hypothesis, $h(G, \lambda_k, \mathcal{F}) = 0$. At this step, the only possible application rules are $R1$ and $R2$. By definition, such rules do not introduce illegitimate configurations, then $h(G, \lambda_{k+1}, \mathcal{F}) = 0$. Therefore, all the labeled graphs (G, λ') obtained from (G, λ) verify the property. Formally, if $h(G, \lambda, \mathcal{F}) = 0$ then $\forall\ (G, \lambda')$, $(G, \lambda) \xrightarrow[\Re]{*} (G, \lambda')$: $h(G, \lambda', \mathcal{F}) = 0$. □

The proof of termination presented in [1] is based on the fact that no state can occur twice in the same run of the protocol. In the self-stabilization context when a system is subject to corruption, this fact is not automatically satisfied. Recall that [1] proposed an extension of its algorithm to deal with any graphs (including ambiguous graphs). Therefore, this extension may be used to treat the case of corruption [2]. Another way consists to treat locally the corruptions, then the corrections' actions are executed during the execution of the enumeration algorithm. Our solution satisfies the last way, its correction is shown in the following Lemma.

Lemma 2. *The system $\Re_s = (L, P_s, \mathcal{F})$ satisfies the convergence property.*

Proof. We study the case of each illegitimate configuration.

1. $\exists\ v_0 \in V(G)$ labeled such as:$n(v_0) \neq 1 + max\{n \mid (n, LV) \in GV(v_0)\}$ or $n(v_0) >\mid V(G) \mid$. The corrupted node v_0 applies rule $RC1$ to change its name to -1, then $RC2$ is executed by one of its neighbors v_i, after that v_0 executes $RC3$ to choose an unused number to rename itself. Let v_i one of its neighbors. The new state of v_i is such that the name (resp. the local view) of v_0 does not appear in the local view (resp. in the global view) of v_i. Then, the *Transmitting rule* allows to diffuse this novel state in all the graph.

2. $\exists\ v_0 \in V(G)$ labeled such as: $\exists\ n_1 \in LV(v_0) \mid \neg\exists\ v_i \in B(v_0)\backslash\{v_0\} : n(v_i) = n_1$ or $n_1 >\mid V(G)\mid$. Correction rule $RC2$ detects and eliminates such configuration.
3. $\exists\ v_0 \in V(G)$ labeled such as: $\exists\ (n_1, LV_1) \in GV(v_0)$, $\delta_{LV_1}(n(v_0)) \geq 2$ or $n_1 >\mid V(G)\mid$. Also, with the same reasoning, rule $RC4$ is applied to detect and eliminate this corrupted configuration.

The size of the relabeling sequence required, to eliminate all the illegitimate configurations and also to terminate the execution of the algorithm, is given in the section related to the complexity analysis. Formally, we have the following properties:

- The application of a correction rule decreases $h(G, \lambda, \mathcal{F})$ and induces the execution of the rule $R1$.
- The application of a rule in P does not increase $h(G, \lambda, \mathcal{F})$.
- $\forall\ (G, \lambda) \in \mathcal{G}_{\mathcal{L}}, \exists$ an integer k,
  $(G, \lambda) \xrightarrow[\Re]{k} (G, \lambda') : h(G, \lambda', \mathcal{F}) = 0$

□

Lemma 3. *The system $\Re_s = (L, P_s, \mathcal{F})$ encodes an enumeration algorithm.*

Proof. To show that the result is an enumeration, we use the same properties as those used in [1, 2]. Let G be a graph, to explain how this protocol denoted by $\mathcal{P}$ works, we introduce some notations. We denote by σ the state of the network which is composed of the local configurations of all the nodes. If $\sigma(v) = (n, LV, GV)$, then, n, LV, GV refer to the name, the local view and the mailbox of v at state σ. Thus, $\gamma(\sigma(v))$ is the name of the node v at state σ. Elements of mailbox are called messages; message m is sent by na if $m = (na, LV)$ for some LV. Name na is *known* to node v at state σ, if $\sigma(v) = (n, LV, GV)$ for some n, LV, GV and GV contains a message sent by na. We denote by $\sigma_k = (n_k(v), LV_k(v), GV_k(v))$ the label of each node $v \in V(G)$ after the k^{th} step of the computation of $\mathcal{P}$. Protocol $\mathcal{P}$ satisfies the following properties whose proofs can follow the scheme of the one used in [1].

(I1) $\forall\ k \geq 0$, $v \in V(G)$: $LV_k(v) = LV_k(B(v)\backslash\{v\}) - \{0\}$,
(I2) $\forall\ k \geq 0$, $v \in V(G)$: $u, w \in B(v)$: $n_k(u) = n_k(w) \Rightarrow u = w$,
(I3) $\forall\ k \geq 0$, $v \in V(G)$: $n_k(v) \leq n_{k+1}(v)$ and $LV_k(v) \preceq LV_{k+1}(v)$ and $GV_k(v) \subseteq GV_{k+1}(v)$,
(I4) $\forall\ k \geq 0$, $v \in V(G)$, $\forall\ (na, LV) \in GV_k(v)$: $\exists\ u \in V(G)$, $n_k(u) = na$,
(I5) $\forall\ k \geq 0$, $v \in V(G)$, $n_k(v) \neq 0$: $\forall\ na, na'$, $na \leq na'$, v known $na' \Rightarrow v$ known na,
(I6) Protocol $\mathcal{P}$ is terminating for any graph,
(I7) The result of any run of $\mathcal{P}$ for any graph G is locally bijective,
(I8) $\mathcal{P}$ is an enumeration protocol for any unambiguous graph.

□

From the proofs of the three previous lemmas, we can state:

Corollary 1. *Starting from any labeled graphs, the system* $\Re_s = (L, P_s, \mathcal{F})$ *terminates.*

Corollary 2. *The relabeling system* $\Re_s$ *is local stabilizing. It encodes a self-stabilizing enumeration algorithm for any unambiguous graph.*

4.2 Complexity

In this section, we consider the model of distributed system described in Section 2.1 and we compute the complexity of our protocol in terms of relabeling rules or steps. The number of steps when the system does not contain any failure component is denoted by $\mathcal{M}$, the number of steps with failures is denoted by Σ. The complexity of the Mazurkiewicz enumeration algorithm is $\mathcal{M} = \theta(\mid V(G) \mid^3$). The time of stabilization of the algorithm proposed in [2] is $\theta(t\times \mid V(G) \mid^2)$ steps, where t is the sum of the number of nodes and the highest name initially known. We use the following properties to give the stabilization time of our protocol:

1. Each node applies one rule in the set of correction rules to correct itself.
2. The application of the correction rules does not add illegitimate configurations. The application of the rule $RC1$ provokes the application of the rules $RC2$, $RC3$ and so $R1$.
3. In the worst case, for f_1 corruptions of names, f_2 corruptions of local view, f_3 corruptions of global view, the nodes apply $3f_1 + f_2 + f_3$ correction rules. Therefore, the stabilization time is $\theta(5\times \mid V(G) \mid^3)$, when rule $R1$ is applied $\mid V(G) \mid^2$ times.

The worst case corresponds to the case of f_1. Let $f = 3f_1$, then the time of stabilization is $\theta(9f \mid V(G) \mid^2)$. So, the time for the enumeration algorithm subject to f corruptions is $\Sigma = \theta(9f \mid V(G) \mid^2 + (\mid V(G) \mid -f)^3)$. Starting from a configuration without corruptions, we obtain $\theta(\mid V(G) \mid^3)$. See that in [2], the parameter t is unbounded, and in our version f is bounded by $\mid V(G) \mid$.

5 An implementation on the Visidia tool

Visidia [11] is a tool to implement, to simulate, to test and to visualize distributed algorithms. It is motivated by the important theoretical results on the use of graph relabeling systems to encode distributed algorithms and to prove their correctness. Visidia provides a library together with an easy interface to implement distributed algorithms described by means of local computations. The distributed system of Visidia is based on asynchronous message passing model. However, it has been assumed that components of such a system do not fail. The threads representing the processes of the computation are created on the same machine.

A stage of computation [13] in Visidia is carried out after some synchronization, which can be achieved by using probabilistic procedures [14]. The processes

are simulated by Java threads. The high level primitives including the synchronization procedures allows the user to implement local computations.

There are three types of local computations. To implement these local computations in an asynchronous message passing system, a randomized synchronization procedure is associated to each type, which are given in the following:

1. *Rendez-vous* (RV): in a computation step, the labels attached to nodes of K_2 (the complete graph with 2 nodes) are modified according to some rules depending on the labels appearing on K_2.
2. *Local Computation 1* (LC1): in a computation step, the label attached to the center of a ball is modified according to some rules depending on the labels of the ball, labels of the leaves are not modified.
3. *Local Computation 2* (LC2): in a computation step, the labels attached to the center and to the leaves of a ball may be modified according to some rules depending on the labels of the ball.

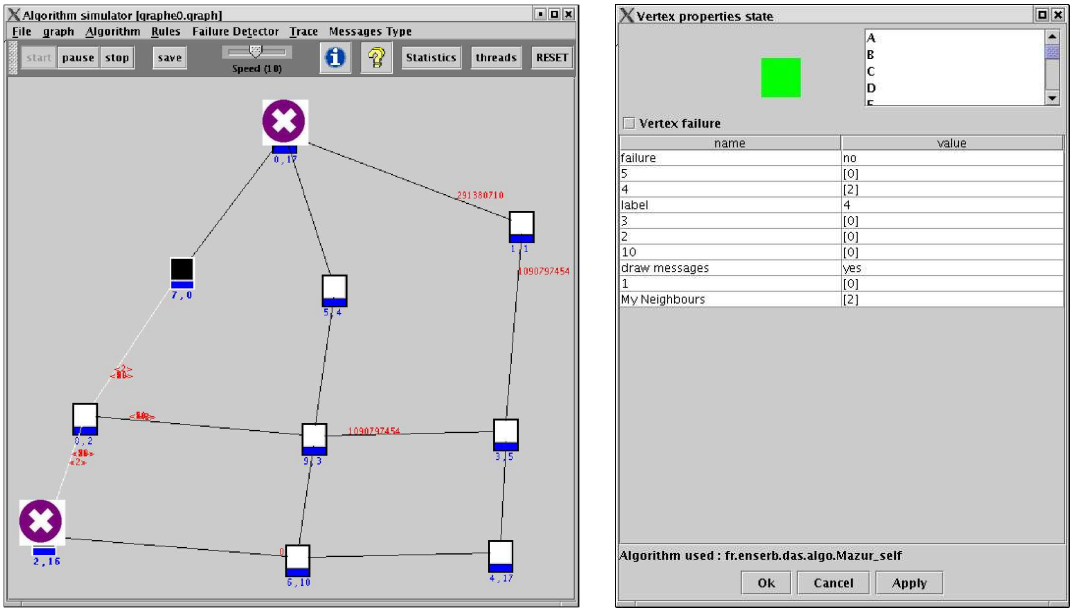

Fig. 1. The beginning of the simulation with transient failures

To simulate transient failures [15], the user can simulate the faulty of a process with the graphical user interface before the beginning of the simulation or during the simulation. For the enumeration algorithm, we show an execution by starting the algorithm with a randomized value of the name (labels). In the following figures, we present an execution of a local self-stabilizing enumeration

algorithm on Visidia starting with faulty values as shown in the left part of Fig.1. For a graph of 10 nodes, each node is represented by two labels $(number, name)$ where *number* is a number used to indicate and distinguish the nodes on the graphical interface and *name* is a value used by the enumeration algorithm. The maximum chosen value doesn't upper to 10, there are two nodes numbered 0 and 2 with incorrect names respectively 17 and 16. In the right part of Fig.1 we show the local view of the node 7, its number is 4 and has a neighbor 8 named 2. In the left part of Fig.2, the faulty node 2 (resp. 0) correct himself to 8 (resp. to 7). Then, the local view of the node 7 contains the correct named neighbor 0. Finally, Fig.3 shows the end of the execution of the enumeration algorithm. In the left part of this figure, the graph is totally named and in the right part we show the final local view of the node 7.

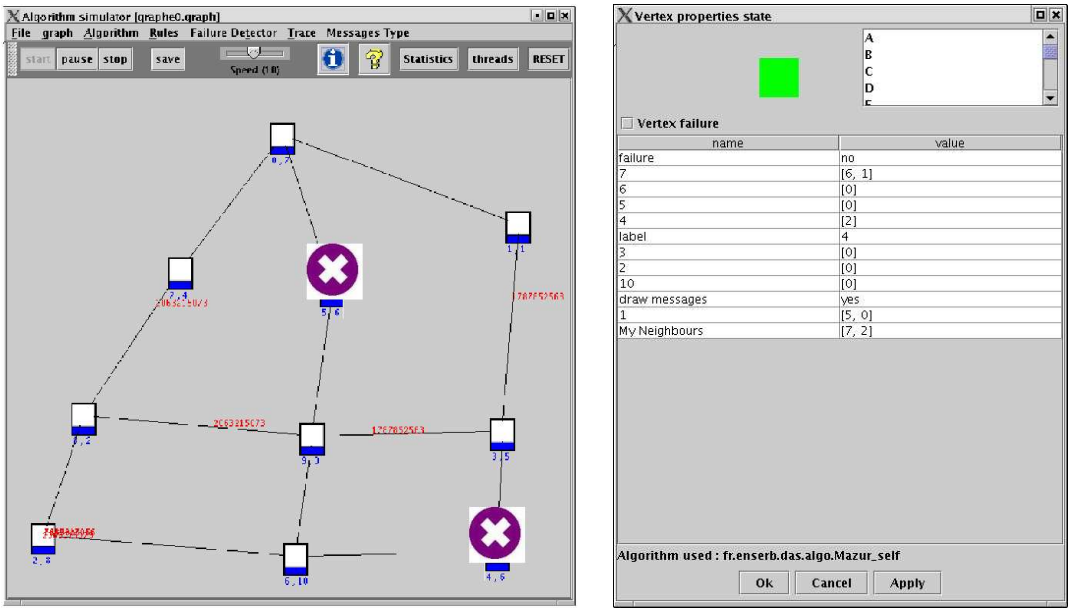

Fig. 2. Local correction of the transient failures

6 Conclusion

In this paper, we have presented a method to encode self-stabilizing enumeration algorithm with local computations. We have adapted the method given in [6] to create an easy self-stabilizing Mazurkiewicz's enumeration algorithm. This kind of algorithm can be used to implement a system which tolerates transient failures. The method is based on defining a set of illegitimate configurations and adding

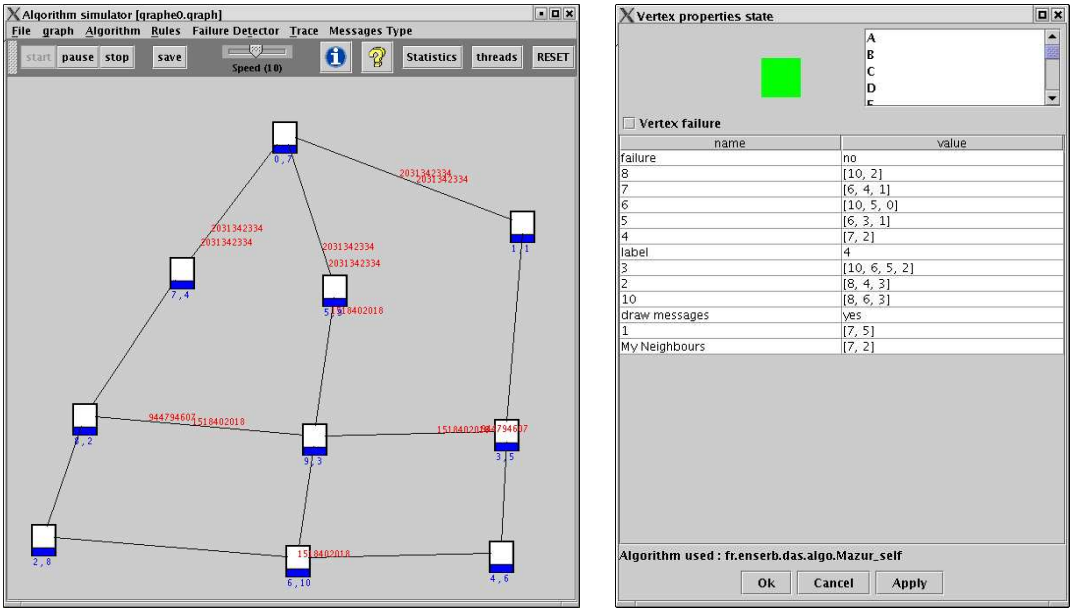

Fig. 3. The end of enumeration and the local views

correction rules to the initial graph rewriting systems. The resulting protocol encoded by local computations is able to detect and correct transient failures by applying correction rules.

This protocol is easy to understand and its translation from the initial algorithm requires little changes. The proof is decomposed into two steps. First the proof of self-stabilization which is based on our developed framework. Second the proof that this protocol does its expected task which is based on the same as [1]. For the complexity study, we show that our protocol is better than [2]. In this work, we had also shown that self-stabilization meets global detection of termination such as [16].

The simulation phase allows us to prove and to show the convergence of our protocol in the presence of transient failures. We show this by starting the execution of the algorithm with faulty labels. Therefore, the system detects and corrects these transient failures by applying correction rules.

References

1. A.W. Mazurkiewicz. Distributed enumeration. *Inf. Processing Letters*, 61(5):233–239, 1997.
2. E. Godard. A self-stabilizing enumeration algorithm. *Inf. Process. Lett.*, 82(6):299–305, 2002.
3. S. Dolev. *Self-stabilization.* MIT Press, 2000.

4. M. Schneider. Self-stabilization. *ACM Computing Surveys*, 25(1):45–67, 1993.
5. E.W. Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.
6. B. Hamid and M. Mosbah. An automatic approach to self-stabilization. In *6th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD2005), Baltimore, USA*, pages 129–132, May 2005.
7. A. Cournier, A.K. Datta, F. Petit, and V. Villain. Self-stabilizing pif algorithms in arbitrary network. *21th International Conference on Distributed Computing Systems (ICDCS 2001)*, pages 91–98, April 2001.
8. S. Ghosh, A. Gupta, T. Herman, and S. V. Pemmaraju. Fault-containing self-stabilizing algorithms. In *PODC '96: Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, pages 45–54. ACM Press, 1996.
9. B. Awerbuch, B. Patt-Shamir, and G. Varghese. Self-stabilization by local checking and correction (extended abstract). In *Proceedings of the 32nd annual symposium on Foundations of computer science*, pages 268–277. IEEE Computer Society Press, 1991.
10. Y. Afek and S. Dolev. Local stabilizer. In *ISTCS '97: Proceedings of the Fifth Israel Symposium on the Theory of Computing Systems (ISTCS '97)*, page 74. IEEE Computer Society, 1997.
11. Y. Afek, S. Kutten, and M. Yung. The local detection paradigm and its applications to self-stabilization. *Theor. Comput. Sci.*, 186(1-2):199–229, 1997.
12. M. Mosbah and A. Sellami. Visidia: A tool for the VIsialization and SImulation of DIstributed Algorithms. http://www.labri.fr/visidia/.
13. I. Litovsky, Y. Métivier, and E. Sopena. Graph relabeling systems and distributed algorithms. In World Scientific Publishing, editor, *Handbook of graph grammars and computing by graph transformation*, volume Vol. III, Eds. H. Ehrig, H.J. Kreowski, U. Montanari and G. Rozenberg, pages 1–56, 1999.
14. M. Bauderon, Y. Métivier, M. Mosbah, and A. Sellami. Graph relabeling systems : A tool for encoding, proving, studying and visualizing distributed algorithms. *Electronic Notes in Theoretical Computer Science*, 51, 2001.
15. Y. Métivier, N. Saheb, and A. Zemmari. Randomized rendezvous. In Birkhauser, editor, *Colloquium on mathematics and computer science: algorithms, trees, combinatorics and probabilities*, Trends in mathematics, pages 183–194, 2000.
16. B. Hamid and M. Mosbah. Visualization of self-stabilizing distributed algorithms. In *9th International conference information visualization IV 2005, London, UK (to appear)*, page .. IEEE Computer Society, 2005.
17. A. Arora and M. Nesterenko. Unifying stabilization and termination in message-passing systems. In *ICDCS '01: Proceedings of the The 21st International Conference on Distributed Computing Systems*, page 99, Washington, DC, USA, 2001. IEEE Computer Society.