Self-Stabilizing Robots in Highly Dynamic Environments

Marjorie Bournat, Ajoy K. Datta, Swan Dubois

Self-Stabilizing Robots in Highly Dynamic Environments*

Marjorie Bournat† Ajoy K. Datta‡ Swan Dubois†

Abstract

This paper deals with the classical problem of exploring a ring by a cohort of synchronous robots. We focus on the perpetual version of this problem in which it is required that each node of the ring is visited by a robot infinitely often.

The challenge in this paper is twofold. First, we assume that the robots evolve in a highly dynamic ring, *i.e.,* edges may appear and disappear unpredictably without any recurrence nor periodicity assumption. The only assumption we made is that each node is infinitely often reachable from any other node. Second, we aim at providing a self-stabilizing algorithm to the robots, *i.e.,* the algorithm must guarantee an eventual correct behavior regardless of the initial state and positions of the robots.

Our main contribution is to show that this problem is deterministically solvable in this harsh environment by providing a self-stabilizing algorithm for three robots.

†UPMC Sorbonne Universités, CNRS, Inria, LIP6 UMR 7606, France

‡University of Nevada, Las Vegas, United States

*This work has been partially supported by the ANR project ESTATE and was initiated while the second author was visiting professor at UPMC Sorbonne Universités.

1 Introduction

We consider a cohort of autonomous and synchronous robots that are equipped with motion actuators and sensors, but that are otherwise unable to communicate [1]. They evolve in a *discrete environment*, where the space is partitioned into a finite number of locations, represented by a graph, where the nodes represent the possible locations of robots and the edges the possibility for a robot to move from one location to another. Refer to [2] for a survey of results in this model. One fundamental problem is the *exploration* of graphs by robots. Basically, each node of the graph has to be visited by at least one robot. There exist several variants of this problem depending on whether the robots are required to stop once they completed the exploration of the graph or not.

Typically, the environment of the robots is modeled by a *static* undirected connected graph where vertices are possible locations of robots and edges represent the moving abilities of the robots. Clearly, such modeling is not suitable for dynamic environments that we use in this paper. Numerous models dealing with topological changes over time have been proposed in the past few decades. There have been some attempts to unifying them as well. The *evolving graphs* were introduced in [3]. They proposed modeling the time as a sequence of discrete time instants and the dynamicity of the system by a sequence of static graphs, one for each instant of time. More recently, another graph model, called *Time-Varying Graphs* (TVG), has been introduced in [4]. In contrast with evolving graphs, TVGs allow systems evolving in continuous time. Also in [4], TVGs are ordered into classes based on mainly two features: the quality of connectivity of the graph and the possibility/impossibility to perform tasks.

As in other distributed systems, *fault-tolerance* is a central issue in robot networks. Indeed, it is desirable that the misbehavior of some robots does not prevent the whole system to reach its objective. *Self-stabilization* [5, 6, 7] is a versatile technique to tolerate *transient* (*i.e.,* of finite duration) faults. After the occurrence of a catastrophic failure that may take the system to some arbitrary global state, self-stabilization guarantees recovery to a correct behavior in finite time without external (*i.e.,* human) intervention. In the context of robot networks, that implies that the algorithm must guarantee an eventual correct behavior regardless of the initial state and positions of the robots.

Our objective in this paper is to study the feasibility of the exploration of a highly dynamic graph by a cohort of self-stabilizing deterministic robots.

Related Work. Since the seminal work of Shannon [8], exploration of graphs by a cohort of robots has been extensively studied. There exist mainly three variants of the problem: (*i*) *exploration with stop*, where robots are required to detect the end of the exploration, then stop moving (*e.g.,* [9]); (*ii*) *exploration with return*, where robots must come back to their initial location once the exploration completed (*e.g.,* [10]); and (*iii*) *perpetual exploration*, where each node has to be infinitely often visited by some robots (*e.g.,* [11]). Even if we restrict ourselves to deterministic approaches, there exist numerous solutions to these problems depending on the topology of the graphs to explore (*e.g.,* ring-shaped [9], line-shaped [12], tree-shaped [13], or arbitrary network [14]), and the assumptions made on robots (*e.g.,* limited range of visibility [15], common sense of orientation [16], *etc.*). But, most of the above work considered only static graphs.

Recently, some work dealt with the exploration of dynamic graphs. The first two papers [17, 18] focused on the exploration (with stop) of so-called periodically varying graphs (*i.e.,* the presence of each edge of the graph is totally periodic). The papers [19, 20, 21] considered another restriction

on dynamicity by considering T-interval-connected graphs (*i.e.,* the graph is always connected and there exists a stability of this connectivity in any interval of time of length T [22]). However, there exist no exploration algorithms for highly dynamic graphs, *i.e.,* graphs where edges may appear and disappear unpredictably without any recurrence, periodicity, or stability assumption and where the only assumption made is that each node is infinitely often reachable from any other node.

To the best of our knowledge, there exist no self-stabilizing algorithm for exploration either in a static or a dynamic environment. Note that there exist solutions in static graphs to other problems (*e.g.,* naming and leader election [23]).

Our Contribution. The main contribution of this paper is to give a positive answer to the open question whether self-stabilizing deterministic exploration of highly dynamic graphs is possible or not. We answer that question by providing a self-stabilizing algorithm to perpetually explore any highly dynamic ring with three deterministic synchronous robots. This is the first exploration algorithm that deals with highly dynamic graphs. This is also the first self-stabilizing algorithm for exploration.

Organization of the paper. This paper is organized as follows. In Section 2, we present the formal model and state the assumptions made. In Section 3, we describe our algorithm. Section 4 contains the proof sketch of our algorithm.

2 Model

In this section, we propose an extension of the classical model of robot networks in static graphs introduced by [24] to the context of dynamic graphs.

Dynamic graphs. In this paper, we consider the model of *evolving graphs* introduced in [3]. We hence consider the time as discretized and mapped to $\mathbb{N}$. An evolving graph $\mathcal{G}$ is an ordered sequence $\{G_1, G_2, \ldots\}$ of subgraphs of a given static graph $G = (V, E)$. In the following, we restrict ourselves to bidirectional graphs. For any $i \geq 0$, we have $G_i = (V, E_i)$ and we say that the edges of E_i are *present* in $\mathcal{G}$ at time i. The *underlying graph* of $\mathcal{G}$, denoted $U_{\mathcal{G}}$, is the static graph gathering all edges that are present at least once in $\mathcal{G}$ (*i.e.,* $U_{\mathcal{G}} = (V, E_{\mathcal{G}})$ with $E_{\mathcal{G}} = \bigcup_{i=0}^{\infty} E_i$). An *eventual missing edge* is an edge of $E_{\mathcal{G}}$ such that there exists a time after which this edge is never present in $\mathcal{G}$. A *recurrent edge* is an edge of $E_{\mathcal{G}}$ that is not eventually missing. The *eventual underlying graph* of $\mathcal{G}$, denoted $U^{\omega}_{\mathcal{G}}$, is the static graph gathering all recurrent edges of $\mathcal{G}$ (*i.e.,* $U^{\omega}_{\mathcal{G}} = (V, E^{\omega}_{\mathcal{G}})$ where $E^{\omega}_{\mathcal{G}}$ is the set of recurrent edges of $\mathcal{G}$). In this paper, we chose to make minimal assumptions on the dynamicity of our graph since we restrict ourselves on *connected-over-time* evolving graphs. The only constraint we impose on evolving graphs of this class is that their eventual underlying graph is connected [25] (intuitively, that means that any node is infinitely often reachable from any other one). For the sake of the proof, we also consider the weaker class of *edge-recurrent* evolving graphs where the eventual underlying graph is connected and matches to the underlying graph. In the following, we consider only connected-over-time evolving graphs whose underlying graph is an anonymous and unoriented ring of arbitrary size. Although the ring is unoriented, to simplify the presentation and discussion, in this paper, we, as external observers, distinguish between the clockwise and the counter-clockwise (global) direction in the ring.

Robots. We consider systems of autonomous mobile entities called robots moving in a discrete and dynamic environment modeled by an evolving graph $\mathcal{G} = \{(V, E_1), (V, E_2) \ldots\}$, V being a set of nodes representing the set of locations where robots may be, E_i being the set of bidirectional edges representing connections through which robots may move from a location to another one at time i. Robots are uniform (they execute the same algorithm), identified (each of them has a distinct identifier), have a persistent memory but are unable to directly communicate with one another by any means. Robots are endowed with local strong multiplicity detection (*i.e.,* they are able to detect the exact number of robots located on their current node). They have no a priori knowledge about the ring they explore (size, diameter, dynamicity. . .). Finally, each robot has its own stable chirality (*i.e.,* each robot is able to locally label the two ports of its current node with *left* and *right* consistently over the ring and time but two different robots may not agree on this labeling). We assume that each robot has a variable *dir* that stores a direction (either *left* or *right*). At any time, we say that a robot points to *left* (resp. *right*) if its *dir* variable is equal to this (local) direction. We say that a robot considers the clockwise (resp., counter-clockwise) direction if the (local) direction pointed to by this robot corresponds to the (global) direction seen by an external observer.

Execution. A configuration γ of the system captures the position (*i.e.,* the node where the robot is currently located) and the state (*i.e.,* the value of every variable of the robot) of each robot at a given time. Given an evolving graph $\mathcal{G} = \{G_1, G_2, \ldots\}$, an algorithm $\mathcal{A}$, and an initial configuration γ_0, the execution $\mathcal{E}$ of $\mathcal{A}$ on $\mathcal{G}$ starting from γ_0 is the infinite sequence $(G_0, \gamma_0), (G_1, \gamma_1), (G_2, \gamma_2), \ldots$ where, for any $i \geq 0$, the configuration γ_{i+1} is the result of the execution of a synchronous round by all robots from (G_i, γ_i) as explained below.

The round that transitions the system from (G_i, γ_i) to (G_{i+1}, γ_{i+1}) is composed of three atomic and synchronous phases: Look, Compute, Move. During the Look phase, each robot gathers information about its environment in G_i. More precisely, each robot updates the value of the following local predicates: (i) $NumberOfRobotsOnNode()$ returns the exact number of robots present at the node of the robot; (ii) $ExistsEdgeOnCurrentDirection()$ returns true if an edge is present at the direction currently pointed by the robot, false otherwise; (iii) $ExistsEdgeOnOppositeDirection()$ returns true if an edge is present in the direction opposite to the one currently pointed by the robot, false otherwise; (iv) $ExistsAdjacentEdge()$ returns true if an edge adjacent to the current node of the robot is present, false otherwise. During the Compute phase, each robot executes the algorithm $\mathcal{A}$ that may modify some of its variables (in particular *dir*) depending on of its current state and the values of the predicates updated during the Look phase. Finally, the Move phase consists of moving each robot trough one edge in the direction it points to if there exists an edge in that direction, otherwise, *i.e.,* if the edge is missing at that time, the robot remains at its current node. Note that the i^{th} round is entirely executed on G_i and that the transition from G_i to G_{i+1} occurs only at the end of this round. We say that a robot is *edge-activated* during a round if there exists at least one edge adjacent to its location during that round.

Self-Stabilization. Intuitively, a self-stabilizing algorithm is able to recover in a finite time a correct behavior from any arbitrary initial configuration (that captures the effect of an arbitrary transient fault in the system). More formally, an algorithm $\mathcal{A}$ is *self-stabilizing* for a problem on a class of evolving graphs $\mathcal{C}$ if and only if it ensures that, for any configuration γ_0, the execution of $\mathcal{A}$ on any $\mathcal{G} \in \mathcal{C}$ starting from γ_0 contains a configuration γ_i such that the execution of $\mathcal{A}$ on $\mathcal{G}$ starting

from γ_i satisfies the specification of the problem. Note that, in the context of robot networks, this definition implies that robots must tolerate both arbitrary initialization of their variables and arbitrary initial positions (in particular, robots may be stacked in the initial configuration).

Perpetual Exploration. Given an evolving graph $\mathcal{G}$, a perpetual exploration algorithm guarantees that every node of $\mathcal{G}$ is infinitely often visited by at least one robot (*i.e.,* a robot is infinitely often located at every node of $\mathcal{G}$). Note that this specification does not require that every robot visits infinitely often every node of $\mathcal{G}$.

3 Exploring a Highly Dynamic Ring with Three Robots

In this section, we present our self-stabilizing deterministic algorithm for the perpetual exploration of any connected-over-time ring with three robots. In this context, the difficulty to complete the exploration is twofold. First, in connected-over-time graphs, robots must deal with the possible existence of some eventual missing edge (without the guarantee that such edge always exists). Note that, in the case of a ring, there is at most one eventual missing edge in any execution (otherwise, we have a contradiction with the connected-over-time property). Second, robots have to handle the arbitrary initialization of the system (corruption of variables and arbitrary position of robots).

Principle of the algorithm. The main idea behind our algorithm is that a robot does not change its direction (arbitrarily initialized) while it is isolated. This allows robots to perpetually explore connected-over-time rings with no eventual missing edge regardless of the initial direction of the robots.

Obviously, this idea is no longer sufficient when there exists an eventual missing edge since, in this case, at least two robots will eventually be stuck (*i.e.,* they point to an eventual missing edge that they are never able to cross) forever at one end of the eventual missing edge. When two (or more) robots are located at the same node, we say that they form a tower. In this case, our algorithm succeed (as we explain below) to ensure that at least one robot leaves the tower in a finite time. In this way, we obtain that, in a finite time, a robot is stuck at each end of the eventual missing edge. These two robots located at two ends of the eventual missing edge play the role of "sentinels" while the third one (we call it a "visitor") visits other nodes of the ring in the following way. The "visitor" keeps its direction until it meets one of these "sentinels", they then switch their roles: After the meeting, the "visitor" still maintains the same direction (becoming thus a "sentinel") while the "sentinel" robot changes its direction (becoming thus a "visitor" until reaching the other "sentinel").

In fact, robots are never aware if they are actually stuck at an eventual missing edge or are just temporarily stuck on an edge that will reappear in a finite time. That is why it is important that the robots keep consider their directions and try to move forward while there is no meeting in order to track a possible eventual missing edge. Our algorithm only guarantees a convergence in a finite time towards a configuration where a robot plays the role of "sentinel" at each end of the eventual missing edge if such an edge exists. Note that, in the case where there is no eventual missing edge, this mechanism does not prevent the correct exploration of the ring since it is impossible for a robot to be stuck forever.

Our algorithm easily deals with the initial corruption of its variables. Indeed, we use variables only to save some information about the environment of the robots in the previous rounds and

we update them at each round. Thus, their arbitrary initial value is erased in a finite time. The main difficulty to achieve self-stabilization is to deal with the arbitrary initial position of robots. In particular, the robots may initially form towers. In the worst case, all robots of a tower may be stuck at an eventual missing edge and be in the same state. They are then unable to start the "sentinels"/"visitor" scheme explained above. Our algorithm needs to "break" such a tower in a finite time (*i.e.,* one robot must leave the node where the tower is located). In other words, we tackle a classical problem of symmetry breaking. We succeed by providing each robot with a function that returns, in a finite number of invocations, different global directions to two robots of the tower based on the private identifier of the robot and without any communication among the robots. More precisely, this is done thanks to a transformation of the robot identifier: each bit of the binary representation of the identifier is duplicated and we add the bits "01" at the end of the sequence of these duplicated bits. Then, at each invocation of the function, a robot reads the next bit of this transformed identifier. If the robot reads zero, it try to move to its left. Otherwise, it try to move to its right. Doing so, in a finite number of invocation of this function, at least one robot leaves the tower. If necessary, we repeat this "tower breaking" scheme until we are able to start the "sentinels"/"visitor" scheme.

The main difficulty in designing this algorithm is to ensure that these two mechanisms ("sentinels"/"visitor" and "tower breaking") do not interfere with each other and prevent the correct exploration. We solve this problem by adding some waiting at good time, especially before starting the procedure of tower breaking by identifier to ensure that robots do not prematurely turn back and "forget" to explore some parts of the ring.

Formal presentation of the algorithm. Before presenting formally our algorithm, we need to introduce the set of constants (*i.e.,* variables assumed to be not corruptible) and the set of variables of each robot. We also introduce three auxiliary functions.

As stated in the model, each robot has an unique identifier. We denote it by id and represent it in binary as $b_0 b_1 \dots b_{|id|-1}$. We define, for the purpose of the "breaking tower" scheme, the constant $TransformedIdentifier$ by its binary representation $b_0 b_0 b_1 b_1 \dots b_{|id|-1} b_{|id|-1} 01$ (each bit of id is duplicated and we add the two bits 01 at the end). We store the length of the binary representation of $TransformedIdentifier$ in the constant ℓ and we denote its ith bit by $TransformedIdentifier[i]$ for any $0 \leq i \leq \ell - 1$.

In addition to the variable dir defined in the model, each robot has the following three variables: (i) the variable $i \in \mathbb{N}$ corresponds to an index to store the position of the last bit read from $TransformedIdentifier$; (ii) the variable $NumberRobotsPreviousEdgeActivation \in \mathbb{N}$ stores the number of robots that were present at the node of the robot during the look step of the last round where it was edge-activated; and (iii) the variable $HasMovedPreviousEdgeActivation \in \{true, false\}$ indicates if the robot has crossed an edge during its last edge-activation.

Our algorithm makes use of a function Update that updates the value of the two last variables according to the current environment of the robot each time it is edge-activated. We provide the pseudo-code of this function in Algorithm 1. Note that this function also allows us to deal with the initial corruption of the two last variables since it resets them in the first round where the robot is edge-activated.

We already stated that, whenever robots are stuck forming a tower, they make use of a function to "break" the tower in a finite time. The pseudo-code of this function GiveDirection appears in Algorithm 2. It assigns the value $left$ or $right$ to the variable dir of the robot depending on

Algorithm 1 Function Update

```
1: function UPDATE
2:     if ExistsAdjacentEdge() then
3:         NumberRobotsPreviousEdgeActivation ← NumberOfRobotsOnNode()
4:         HasMovedPreviousEdgeActivation ← ExistsEdgeOnCurrentDirection()
5:     end if
6: end function
```

the the ith bit of the value of $TransformedIdentifier$. The variable i is incremented modulo ℓ (that implicitly resets this variable when it is corrupted) to ensure that successive calls to GiveDirection will consider each bit of $TransformedIdentifier$ in a round-robin way. As shown in the next section, this function guarantees that, if two robots are stuck together in a tower and invoke repeatedly their own function GiveDirection, then two distinct global directions are given in finite time to the two robots regardless of their chirality. This property allows the algorithm to "break" the tower since at least one robot is then able to leave the node where the tower is located.

Finally, we define the function OppositeDirection that simply affects the value $left$ (resp. $right$) to the variable dir when $dir = right$ (resp. $dir = left$).

There are two types of configurations in which the robots may change the direction they consider. So, our algorithm needs to identify them. We do so by defining a predicate that characterizes each of these configurations.

The first one, called $WeAreStuckInTheSameDirection()$, is dedicated to the detection of configurations in which the robot must invoke the "tower breaking" mechanism. Namely, the robot is stuck since at least one edge-activation with at least another robot and the edge in the direction opposite to the one considered by the robot is present. More formally, this predicate is defined as follows:

$$\begin{aligned} &WeAreStuckInTheSameDirection() \equiv \\ &\quad (NumberOfRobotsOnNode() > 1) \\ &\wedge\ (NumberOfRobotsOnNode() = NumberRobotsPreviousEdgeActivation) \\ &\wedge\ \neg ExistsEdgeOnCurrentDirection() \\ &\wedge\ ExistsEdgeOnOppositeDirection() \\ &\wedge\ \neg HasMovedPreviousEdgeActivation \end{aligned}$$

The second predicate, called $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$, isdesigned to detect configurations in which the robot must transition from the "sentinel" to the "visitor" role in the "sentinel"/"visitor" scheme. More precisely, such configuration is characterized by the fact that the robot is edge-activated, stuck during its previous edge-activation, and there are strictly more robots located at its node than at its previous edge-activation. More formally, this predicate is defined as follows:

$$\begin{aligned} &IWasStuckOnMyNodeAndNowWeAreMoreRobots() \equiv \\ &\quad (NumberOfRobotsOnNode() > NumberRobotsPreviousEdgeActivation) \\ &\wedge \neg HasMovedPreviousEdgeActivation \\ &\wedge\ ExistsAdjacentEdge() \end{aligned}$$

Now, we are ready to present the pseudo-code of the core of our algorithm (see Algorithm 3). The basic idea of the algorithm is the following. The function GiveDirection is invoked when $WeAreStuckInTheSameDirection()$ is true (to try to "break" the tower after the appropriate waiting), while the function OppositeDirection is called when $IWasStuckOnMyNodeAndNowWe-$

Algorithm 2 Function GiveDirection

```
1: function GiveDirection
2:     i ← i + 1 (mod ℓ)
3:     if TransformedIdentifier[i] = 0 then
4:         dir ← left
5:     else
6:         dir ← right
7:     end if
8: end function
```

Algorithm 3 Self-stabilizing perpetual exploration

```
1: if WeAreStuckInTheSameDirection() then
2:     GiveDirection
3: end if
4: if IWasStuckOnMyNodeAndNowWeAreMoreRobots() then
5:     OppositeDirection
6: end if
7: Update
```

$AreMoreRobots()$ is true (to implement the "sentinel"/"visitor" scheme). Afterwards, the function Update is called (to update the state of the robot according to its environment).

4 Proof Sketch

Due to the lack of space, we present only a sketch of the proof of our algorithm in this paper. This section captures the main ideas behind the proof and summarizes its main steps. The detailed proof of our algorithm is available in a companion technical report [26].

Preliminaries. First, we introduce some definitions and preliminary results that are extensively used in the proof.

We saw previously that the notion of tower is central in our algorithm. Intuitively, a tower captures the simultaneous presence of all robots of a given set on a node at each time of a given interval. We require either the set of robots or the time interval of each tower to be maximal. Note that the tower is not required to be on the same node at each time of the interval (robots of the tower may move together without leaving the tower).

We distinguish two kinds of towers according to the agreement of their robots on the global direction to consider at each time there exists an adjacent edge to their current location (excluded the last one). If they agreed, the robots form a long-lived tower while they form a short-lived tower in the contrary case. This implies that a short-lived tower is broken as soon as the robots forming the tower are edge-activated, while the robots of a long-lived tower move together at each edge activation of the tower (excluded the last one).

Definition 1 (Tower) *A tower T is a couple (S, θ), where S is a set of robots ($|S| > 1$) and $\theta = [t_s, t_e]$ is an interval of $\mathbb{N}$, such that all the robots of S are located at a same node at each instant of time t in θ and S or θ are maximal for this property. Moreover, if the robots of S move during a round $t \in [t_s, t_e[$, they are required to traverse the same edge.*

Definition 2 (Long-lived tower) *A long-lived tower $T = (S, [t_s, t_e])$ is a tower such that there is at least one edge-activation of all robots of S in the time interval $[t_s, t_e[$.*

Definition 3 (Short-lived tower) *A short-lived tower T is a tower that is not a long-lived tower.*

For $k > 1$, a long-lived (resp., a short-lived) tower $T = (S, \theta)$ with $|S| = k$ is called a k-long-lived (resp., a k-short-lived) tower.

In the remainder of this section, we consider an execution $\mathcal{E}$ of Algorithm 3 executed by three robots r_1, r_2, and r_3 on a connected-over-time ring $\mathcal{G}$ of size $n \in \mathbb{N}^*$ starting from an arbitrary configuration.

For the sake of clarity, the value of a variable or a predicate *name* of a given robot r at the end of the Look phase of a given round t is denoted by the notation $name(r, t)$.

We say that a robot r has a coherent state at time t, if during the Look phase of round t, the value of its variable $NumberRobotsPreviousEdgeActivation(r, t)$ corresponds to the value of its predicate $NumberOfRobotsOnNode()$ at its previous edge-activation and the value of its variable $HasMovedPreviousEdgeActivation(r, t)$ corresponds to the value of its predicate $ExistsEdgeOnCurrentDirection()$ at its previous edge-activation. The following lemmastates that, for each robot, there exists a suffix of the execution in which the robot is coherent.

Lemma 1 *For any robot, there exists a time from which its state is always coherent.*

Let t_1, t_2, and t_3 be respectively the time at which the robot r_1, r_2, and r_3, respectively are in a coherent state. Let $t_{max} = max\{t_1, t_2, t_3\}$. From Lemma 1, the three robots are in a coherent state from t_{max}. In the remaining of the proof, we focus on the suffix of the execution after t_{max}.

The two following lemmas show that, regardless of the chirality of the robots and the initial values of their variables i, a finite number of synchronous invocations of the function GiveDirection by two robots of a tower returns them a distinct global direction. To prove that, we need to take a close look at properties granted by the transformed identifiers of the robots.

Lemma 2 *Let tl_1 and tl_2 be two transformed identifiers, such that $tl_1 \neq tl_2$. Let i and j be two integers such that $i \in [0, |tl_1| - 1]$ and $j \in [0, |tl_2| - 1]$. If $tl_1[i] = tl_2[j]$, then there exists an integer k such that $tl_1[(i + k) \pmod{|tl_1|}] \neq tl_2[(j + k) \pmod{|tl_2|}]$.*

Lemma 3 *Let tl_1 and tl_2 be two transformed identifiers, such that $tl_1 \neq tl_2$. Let i and j be two integers such that $i \in [0, |tl_1| - 1]$ and $j \in [0, |tl_2| - 1]$. If $tl_1[i] \neq tl_2[j]$, then there exists an integer k such that $tl_1[(i + k) \pmod{|tl_1|}] = tl_2[(j + k) \pmod{|tl_2|}]$.*

Technical lemmas on towers. We are now able to state a set of lemmas that show some interesting technical properties of towers under specific assumptions during the execution of our algorithm. These properties are extensively used in the main proof of our algorithm. Their proofs are very technical, hence omitted due to lack of space.

Lemma 4 *The robots of a long-lived tower $T = (S, [t_s, t_e])$ consider a same global direction at each time between the Look phase of round t_s and the Look phase of round t_e included.*

Lemma 5 *If there exists an eventual missing edge, then all long-lived towers have a finite duration.*

Lemma 6 *Every execution containing only configurations without any long-lived tower cannot reach a configuration with a 3-short-lived tower.*

Lemma 7 *Every execution starting from a configuration without a 3-long-lived tower cannot reach a configuration with a 3-long-lived tower.*

Lemma 8 *Let γ be a configuration such that all but one robots consider the same global direction. Then starting from γ, no execution without any long-lived towers can reach a configuration where all robots consider the same global direction.*

Lemma 9 *Consider an execution containing no 3-long-lived towers. If a 2-long-lived tower $T = (S, [t_s, t_e])$ is located at a node u at round t_e, then the robot that does not belong to S cannot be located at node u during the Look phase of round t_e. Moreover during the Look phase of round $t_e + 1$, one robot of S located at u considers a global direction opposite to the one considered by the other robot of S (which is not on u).*

The following lemma is used to prove, in combination with Lemmas 2 and 3, the "tower breaking" mechanism since it proves that robots of a long-lived tower synchronously invoke their GiveDirection function after their first edge-activation.

Lemma 10 *For any long-lived tower $T = (S, [t_s, t_e])$, any (r_i, r_j) in S^2, and any t less or equal to t_e, we have $WeAreStuckInTheSameDirection()(r_i, t) = WeAreStuckInTheSameDirection()(r_j, t)$ if all robots of S have been edge-activated between t_s (included) and t (not included).*

The next two lemmas show that the whole ring is visited between two consecutive 2-long-lived towers if these two towers satisfy some properties. They are used in the proof of the "sentinels"/"visitor" scheme.

Lemma 11 *Consider an execution $\mathcal{E}$ without any 3-long-lived tower but containing a 2-long-lived tower $T = (S, [t_s, t_e])$. If there exists another 2-long-lived tower $T' = (S', [t'_s, t'_e])$ after T in $\mathcal{E}$ and if T' is the first 2-long-lived tower in $\mathcal{E}$ such that $t'_s > t_e + 1$, then all the edges of $\mathcal{G}$ have been crossed by at least one robot between time t_e and time t'_s.*

Lemma 12 *Consider that there are no 3-long-lived towers in $\mathcal{E}$, and let $T_i = (S_i, [t_{s\_i}, t_{e\_i}])$ be the i^{th} 2-long-lived tower of $\mathcal{E}$ (with $i \geq 2$). If $T_{i+1} = (S_{i+1}, [t_{s\_i+1}, t_{e\_i+1}])$ exists such that $t_{s\_i+1} = t_{e\_i} + 1$, then all the edges of $\mathcal{G}$ have been crossed by at least one robot between time $t_{s\_i} - 1$ and time t_{s_i+1}.*

Main lemmas. Upon establishing all the above properties of towers, we are now ready to state the main lemmas of our proof. Each of these three lemmas below shows that our algorithm performs the perpetual exploration in a self-stabilizing way for a specific subclass of connected-over-time rings. We only sketch the proof of these lemmas due to space constraints.

Lemma 13 *Algorithm 3 is a self-stabilizing perpetual exploration algorithm for the class of static rings of arbitrary size using three robots.*

Sketchproof 1 *Assume that $\mathcal{G}$ is a static ring. The robots executing our algorithm consider a direction in each round. Moreover, in our algorithm, the robots do not change the direction they consider if there exists an adjacent edge to their current location in the direction they consider. As $\mathcal{G}$ is static, this implies that in each round all the edges of $\mathcal{G}$ are present. Thus, the robots never change their directions.*

As (i) the robots have a stable direction, (ii) they always consider the same global direction, and (iii) there always exists an adjacent edge to their current locations in the global direction they consider, the robots move infinitely often in the same global direction. Moreover, as $\mathcal{G}$ has a finite size, this implies that all the robots visit infinitely often all the nodes of $\mathcal{G}$.

Lemma 14 *Algorithm 3 is a self-stabilizing perpetual exploration algorithm for the class of edge-recurrent but non static rings of arbitrary size using three robots.*

Sketchproof 2 *Assume that $\mathcal{G}$ is an edge-recurrent but non static ring. Let us study the following cases.*

Case 1: *There exists at least one 3-long-lived tower in $\mathcal{E}$.*

Case 1.1: *One of the 3-long-lived towers of $\mathcal{E}$ has an infinite duration.*

Denote by T the 3-long-lived tower of $\mathcal{E}$ that has an infinite duration. According to the definition of the predicate $WeAreStuckInTheSameDirection()$, Lemmas 10, 2, and 3, T is eventually not stuck during two consecutive edge-activations. Thus, after the formation of T, the robots see infinitely often an adjacent edge in the direction they consider. Moreover, as there are three robots in the system, the predicate $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ is not true for these robots. Thus, the three robots eventually consider the same global direction. This implies that the three robots are able to move infinitely often in the same global direction. Moreover, as $\mathcal{G}$ has a finite size, all the robots visit infinitely often all the nodes of $\mathcal{G}$.

Case 1.2: *Any 3-long-lived tower of $\mathcal{E}$ has a finite duration.*

By Lemma 7, once a 3-long-lived tower is broken, it is impossible to have another 3-long-lived tower in $\mathcal{E}$. Then, $\mathcal{E}$ admits an infinite suffix that matches either case 2 or 3.

Case 2: *There exists at least one 2-long-lived tower in $\mathcal{E}$.*

Case 2.1: *There exists a finite number of 2-long-lived towers in $\mathcal{E}$.*

If the last 2-long-lived tower of $\mathcal{E}$ has a finite duration, then $\mathcal{E}$ admits an infinite suffix with no long-lived towers thus matching Case 3.

*Otherwise, (*i.e., *the last 2-long-lived tower T of $\mathcal{E}$ has an infinite duration), as in Case 1.1, the robots of the 2-long-lived tower eventually see infinitely often an adjacent edge in the direction they consider since they are eventually not stuck. The only case when the robots of the 2-long-lived tower change their direction is when they meet the third robot of the system.*

Case 2.1.1: *The robots of T meet the third robot finitely often.*

After the last meeting with the third robot in the system, the robots of T have their predicates $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ always false. Thus,

after the last meeting, the robots of T always consider the same global direction. This implies that they are able to move infinitely often in the same global direction. Moreover, as $\mathcal{G}$ has a finite size, this implies that all the robots visit infinitely often all the nodes of $\mathcal{G}$.

Case 2.1.2: *The robots of T meet the third robot infinitely often.*

The third robot does not change its direction while it is isolated. Similarly, the robots of T maintain their directions until they meet the third robot. Moreover, we can prove that, after a meeting, the third robot and the robots of T consider two opposite global directions. Then, we can deduce that all the nodes of $\mathcal{G}$ are visited between two consecutive meetings of T and the third robot. As T and the third robot infinitely often meet, the nodes of $\mathcal{G}$ are infinitely often visited.

Case 2.2: *There exist an infinite number of 2-long-lived towers in $\mathcal{E}$.*

By Lemmas 11 and 12, we know that between two consecutive 2-long-lived towers (from the second one), all the edges, and thus all the nodes of $\mathcal{G}$ are visited. As there is an infinite number of 2-long-lived towers, the nodes of $\mathcal{G}$ are infinitely often visited.

Case 3: *There exist no long-lived towers in $\mathcal{E}$.*

Then, we know, by Lemma 6, that $\mathcal{E}$ contains only configurations with either three isolated robots or one 2-short-lived tower and one isolated robot. The robots can then start the "sentinels"/"visiting" scheme which permits the perpetual exploration of $\mathcal{G}$ even when there is no eventual missing edge. Note that, as there is no eventual missing edge, it is possible that no 2-short-lived towers are formed. In this case, all the robots maintain their directions.

Thus, we obtain the desired result in every cases.

Lemma 15 *Algorithm 3 is a self-stabilizing perpetual exploration algorithm for the class of connected-over-time but not edge-recurrent rings of arbitrary size using three robots.*

Sketchproof 3 *Consider a connected-over-time but not edge-recurrent ring. This implies that there exists exactly one eventual missing edge in $\mathcal{E}$. Denote by $\mathcal{E}^1$ the suffix of $\mathcal{E}$ in which the eventual missing edge never appears.*

Assume that there exists a 3-long-lived tower in $\mathcal{E}^1$. According to Lemma 5, this 3-long-liver tower is broken in finite time. Moreover, once this tower is broken, according to Lemma 7, it is impossible to have a configuration containing a 3-long-lived tower. Then, $\mathcal{E}^1$ admits an infinite suffix $\mathcal{E}^2$ without a 3-long-lived tower.

Assume that there exists a 2-long-lived tower in $\mathcal{E}^2$. According to Lemma 5, this 2-long-lived tower is broken in finite time. Once this tower is broken, in the remainder of $\mathcal{E}^2$, there exists at most one 2-long-lived tower T_{second}. Indeed, according to Lemma 5, if T_{second} exists, it is broken in finite time. Moreover, T_{second} cannot be the first 2-long-lived tower of the execution. Then, by Lemmas 11 and 12, $\mathcal{E}^2$ admits an infinite suffix $\mathcal{E}^3$ without a 2-long-lived tower.

In other words, there are no long-lived towers in $\mathcal{E}^3$. By Lemma 6, all configurations in $\mathcal{E}^3$ contain either three isolated robots or one 2-short-lived tower and one isolated robot. The robots can then start the "sentinels"/"visiting" scheme which permits the perpetual exploration of $\mathcal{G}$ and proves the result.

The end of the road. To conclude the proof, it is sufficient to observe that a connected-over-time ring is by definition either static, edge-recurrent but non static, or connected-over-time but not edge-recurrent. As we prove the self-stabilization of our algorithm in these three cases in Lemmas 13, 14, and 15, we can claim the following final result.

Theorem 1 *Algorithm 3 is a self-stabilizing perpetual exploration algorithm for the class of connected-over-time rings of arbitrary size using three robots.*

5 Conclusion

In this paper, we addressed the open question: "Is it possible to achieve self-stabilization for swarm of robots evolving in highly dynamic graphs?". We answered positively to this question by providing a self-stabilizing algorithm for three synchronous robots that perpetually explore any connected-over-time ring, *i.e.,* any dynamic ring with very weak assumption on connectivity: every node is infinitely often reachable from any another one without any recurrence, periodicity, nor stability assumption.

In addition to the above contributions, our algorithm overcomes the robot networks state-of-the-art in a couple of ways. First, it is the first algorithm dealing with highly dynamic graphs. All previous solutions made some assumptions on periodicity or on all-time connectivity of the graph. Second, it is the first self-stabilizing algorithm for the problem of exploration, either for static or for dynamic graphs.

This work opens an interesting field of research with numerous open questions. First, we should investigate the necessity of every assumption made in this paper. For example, we assumed that robots are synchronous. Is this problem solvable with asynchronous robots? Second, we can investigate the issue of the number of robots. What are the minimal/maximal number of robots to solve the problem? It would be worthwhile to explore other problems in this rather complicated environment, *e.g.,* gathering, leader election, *etc.*. It may also be interesting to consider other classes of dynamic graphs and other classes of faults, *e.g.,* crashes of robots, Byzantine failures, *etc.*.

References

[1] Suzuki, I., Yamashita, M.: Distributed anonymous mobile robots: Formation of geometric patterns. SIAM Journal on Computers **28**(4) (1999) 1347–1363

[2] Potop-Butucaru, M., Raynal, M., Tixeuil, S.: Distributed computing with mobile robots: An introductory survey. In: International Conference on Network-Based Information Systems (NBiS). (2011) 318–324

[3] Xuan, B., Ferreira, A., Jarry, A.: Computing shortest, fastest, and foremost journeys in dynamic networks. International Journal of Foundations of Computer Science **14**(02) (2003) 267–285

[4] Casteigts, A., Flocchini, P., Quattrociocchi, W., Santoro, N.: Time-varying graphs and dynamic networks. International Journal of Parallel, Emergent and Distributed Systems **27**(5) (2012) 387–408

[5] Dijkstra, E.: Self-stabilizing systems in spite of distributed control. Communication of the ACM **17**(11) (1974) 643–644

[6] Dolev, S.: Self-stabilization. MIT Press (2000)

[7] Tixeuil, S.: Self-stabilizing Algorithms. Chapman & Hall. In: Algorithms and Theory of Computation Handbook. CRC Press, Taylor & Francis Group (2009) 26.1–26.45

[8] Shannon, C.: Presentation of a maze-solving machine. 8th Conference of the Josiah Macy, Jr. Foundation (1951) 173–180

[9] Flocchini, P., Ilcinkas, D., Pelc, A., Santoro, N.: Computing without communicating: Ring exploration by asynchronous oblivious robots. In: International Conference on Principles of Distributed Systems (OPODIS). (2007) 105–118

[10] Diks, K., Fraigniaud, P., Kranakis, E., Pelc, A.: Tree exploration with little memory. Journal of Algorithms **51**(1) (2004) 38–63

[11] Baldoni, R., Bonnet, F., Milani, A., Raynal, M.: On the solvability of anonymous partial grids exploration by mobile robots. In: International Conference on Principles of Distributed Systems (OPODIS). (2008) 428–445

[12] Flocchini, P., Ilcinkas, D., Pelc, A., Santoro, N.: How many oblivious robots can explore a line. Information Processing Letters **111**(20) (2011) 1027–1031

[13] Flocchini, P., Ilcinkas, D., Pelc, A., Santoro, N.: Remembering without memory: Tree exploration by asynchronous oblivious robots. Theoretical Computer Science **411**(14-15) (2010) 1583–1598

[14] Chalopin, J., Flocchini, P., Mans, B., Santoro, N.: Network exploration by silent and oblivious robots. In: Workshop on Graph-Theoretic Concepts in Computer Science (WG). (2010) 208–219

[15] Datta, A., Lamani, A., Larmore, L., Petit, F.: Ring exploration by oblivious agents with local vision. In: IEEE International Conference on Distributed Computing Systems (ICDCS). (2013) 347–356

[16] Blin, L., Milani, A., Potop-Butucaru, M., Tixeuil, S.: Exclusive perpetual ring exploration without chirality. In: International Symposium on Distributed Computing (DISC). (2010) 312–327

[17] Flocchini, P., Mans, B., Santoro, N.: Exploration of periodically varying graphs. In: International Symposium on Algorithms and Computation (ISAAC). (2009) 534–543

[18] Ilcinkas, D., Wade, A.M.: On the power of waiting when exploring public transportation systems. In: International Conference on Principles of Distributed Systems (OPODIS). (2011) 451–464

[19] Ilcinkas, D., Wade, A.M.: Exploration of the t-interval-connected dynamic graphs: The case of the ring. In: Colloquium on Structural Information & Communication Complexity (SIROCCO). (2013) 13–23

[20] Ilcinkas, D., Klasing, R., Wade, A.M.: Exploration of constantly connected dynamic graphs based on cactuses. In: Colloquium on Structural Information & Communication Complexity (SIROCCO). (2014) 250–262

[21] Di Luna, G., Dobrev, S., Flocchini, P., Santoro, N.: Live exploration of dynamic rings. In: IEEE International Conference on Distributed Computing Systems (ICDCS). (2016) 570–579

[22] Kuhn, F., Lynch, N., Oshman, R.: Distributed computation in dynamic networks. In: Symposium on the Theory of Computing (STOC). (2010) 513–522

[23] Blin, L., Potop-Butucaru, M., Tixeuil, S.: On the self-stabilization of mobile robots in graphs. In: International Conference on Principles of Distributed Systems (OPODIS). (2007) 301–314

[24] Klasing, R., Markou, E., Pelc, A.: Gathering asynchronous oblivious mobile robots in a ring. In: International Symposium on Algorithms and Computation (ISAAC). (2006) 744–753

[25] Dubois, S., Kaaouachi, M., Petit, F.: Enabling minimal dominating set in highly dynamic distributed systems. In: International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS). (2015) 51–66

[26] Bournat, M., Datta, A.K., Dubois, S.: Self-stabilizing robots in highly dynamic environments. Technical report, arXiv:1609.06161 (2016)