Gracefully Degrading Gathering in Dynamic Rings*

Marjorie Bournat, Swan Dubois, and Franck Petit

Sorbonne Université, CNRS, Inria, LIP6, F-75005 Paris, France
firstname.lastname@lip6.fr

Abstract. Gracefully degrading algorithms [Biely et al., TCS 2018] are designed to circumvent impossibility results in dynamic systems by adapting themselves to the dynamics. Indeed, such an algorithm solves a given problem under some dynamics and, moreover, guarantees that a weaker (but related) problem is solved under a higher dynamics under which the original problem is impossible to solve. The underlying intuition is to solve the problem whenever possible but to provide some kind of quality of service if the dynamics become (unpredictably) higher.
In this paper, we apply for the first time this approach to robot networks. We focus on the fundamental problem of gathering a squad of autonomous robots on an unknown location of a dynamic ring. In this goal, we introduce a set of weaker variants of this problem. Motivated by a set of impossibility results related to the dynamics of the ring, we propose a gracefully degrading gathering algorithm.

Keywords: Gracefully degrading algorithm · dynamic ring · gathering

1 Introduction

The classical approach in distributed computing consists in, first, fixing a set of assumptions that captures the properties of the studied system (atomicity, synchrony, faults, communication modalities, etc.) and, then, focusing on the impact of these assumptions in terms of calculability and/or of complexity on a given problem. When coming to dynamic systems, it is natural to adopt the same approach. Many recent works focus on defining pertinent assumptions for capturing the dynamics of those systems [8, 13, 19]. When these assumptions become very weak, that is, when the system becomes highly dynamic, a somewhat frustrating but not very surprising conclusion emerge: many fundamental distributed problems are impossible at least, in their classical form [2, 6, 7].

To circumvent such impossibility results, Biely et al. recently introduced the *gracefully degrading* approach [2]. This approach relies on the definition of weaker but related variants of the considered problem. A gracefully degrading algorithm guarantees that it will solve simultaneously the original problem under some

* Work partly funded by Project ESTATE (Ref. ANR-16-CE25-0009-03), supported by French state funds managed by the ANR (Agence Nationale de la Recherche)

assumption of dynamics and each of its variants under some other (hopefully weaker) assumptions. As an example, Biely et al. provide a consensus algorithm that gracefully degrades to k-set agreement when the dynamics of the system increase. The underlying idea is to solve the problem in its strongest variant when connectivity conditions are sufficient but also to provide (at the opposite of a classical algorithm) some minimal quality of service described by the weaker variants of the problem when those conditions degrade.

Note that, although being applied to dynamic systems by Biely et al. for the first time, this natural idea is not a new one. Indeed, *indulgent* algorithms [1, 14] provide similar graceful degradation of the problem to satisfy with respect to synchrony (not with respect to dynamics). *Speculation* [9, 12] is a related, but somewhat orthogonal, concept. A speculative algorithm solves the problem under some assumptions and moreover provides stronger properties (typically better complexities) whenever conditions are better.

The goal of this paper is to apply graceful degradation to robot networks where a cohort of autonomous robots have to coordinate their actions in order to solve a global task. We focus on *gathering* in a *dynamic ring*. In this problem, starting from any initial position, robots must meet on an arbitrary location in a bounded time (that may depend on any parameter about the robots or the ring). Note that we can classically split this specification into a liveness property (all robots terminate in bounded time) and a safety property (all robots that terminate do so on the same node).

Related works. Several models of dynamic graphs have been defined recently [8, 15, 19]. In this paper, we adopt the *evolving graph* model [19] in which a dynamic graph is simply a sequence of static graphs on a fixed set of nodes: each graph of this sequence contains the edges of the dynamic graph present at a given time. We also consider the hierarchy of dynamics assumptions introduced by Casteigts et al. [8]. The idea behind this hierarchy is to gather all dynamic graphs that share some temporal connectivity properties within classes. This allows us to compare the strength of these temporal connectivity properties based on the inclusion of classes between them. We are interested in the following classes: $\mathcal{COT}$ (*connected-over-time* graphs) where edges may appear and disappear without any recurrence nor periodicity assumption but guaranteeing that each node is infinitely often reachable from any other node; $\mathcal{RE}$ (*recurrent-edge* graphs) where any edge that appears at least once does so recurrently; $\mathcal{BRE}$ (*bounded-recurrent-edge* graphs) where any edge that appears at least once does so recurrently in a bounded time; $\mathcal{AC}$ (*always-connected* graphs) where the graph is connected at each instant; and $\mathcal{ST}$ (*static* graphs) where any edge that appears at least once is always present. Note that $\mathcal{ST} \subset \mathcal{BRE} \subset \mathcal{RE} \subset \mathcal{COT}$ and $\mathcal{ST} \subset \mathcal{AC} \subset \mathcal{COT}$ by definition.

In robot networks, the gathering problem was extensively studied in the context of static graphs, e.g., [10, 11, 18]. The main motivation of this vein of research is to characterize the initial positions of the robots allowing gathering in each studied topology in function of the assumptions on the robots as identifiers, communication, vision range, memory, etc. On the other hand, few algorithms have been designed for robots evolving in dynamic graphs. The majority of them

	$\mathbb{G}$	$\mathbb{G}_E$	$\mathbb{G}_W$	$\mathbb{G}_{EW}$
$\mathcal{COT}$	Impossible (Cor. 2 & 3)	Impossible (Cor. 1)	Impossible (Cor. 3)	Possible (Th. 2)
$\mathcal{AC}$	Impossible (Cor. 2)	Impossible (Th. 1)	Possible (Th. 3)	—
$\mathcal{RE}$	Impossible (Cor. 3)	Possible (Th. 4)	Impossible (Cor. 3)	—
$\mathcal{BRE}$	Possible (Th. 5)	—	—	—
$\mathcal{ST}$	Possible (Cor. 4)	—	—	—

Table 1. Summary of our results. The symbol — means that a stronger variant of the problem is already proved solvable under the dynamics assumption.

deals with the problem of exploration [3, 4, 16] (robots must visit each node of the graph at least once or infinitely often depending on the variant of the problem). In the most related work to ours [17], Di Luna et al. study the gathering problem in dynamic rings. They first note the impossibility of the problem in the $\mathcal{AC}$ class and consequently propose a weaker variant of the problem, the near-gathering: all robots must gather in finite time on two adjacent nodes. They characterize the impact of chirality (ability to agree on a common orientation) and cross-detection (ability to detect whenever a robot cross the same edge in the opposite direction) on the solvability of the problem. All their algorithms are designed for the $\mathcal{AC}$ class and are not gracefully degrading.

Contributions. By contrast with the work of Di Luna et al. [17], we keep unchanged the safety of the classical gathering problem (all robots that terminate do so on the same node) and, to circumvent impossibility results, we weaken only its liveness: at most one robot may not terminate or (not exclusively) all robots that terminate do so eventually (and not in a *bounded* time as in the classical specification). This choice is motivated by the approach of indulgent algorithms [1, 14]: the safety captures the "essence" of the problem and should be preserved even in degraded variants of the problem. Namely, we obtain the four following variants of the gathering problem: $\mathbb{G}$ (*gathering*) all robots terminate on the same node in *bounded* time; $\mathbb{G}_E$ (*eventual gathering*) all robots terminate on the same node in *finite* time; $\mathbb{G}_W$ (*weak gathering*) all robots but (at most) one terminate on the same node in *bounded* time; and $\mathbb{G}_{EW}$ (*eventual weak gathering*) all robots but (at most) one terminate on the same node in *finite* time.

We present then a set of impossibility results, summarized in Table 1, for these specifications for different classes of dynamic rings. Motivated by these impossibility results, our main contribution is a gracefully degrading gathering algorithm. For each class of dynamic rings we consider, our algorithm solves the strongest possible of our variants of the gathering problem (see Table 1). This challenging property is obtained without any knowledge or detection of the dynamics by the robots that always execute the same algorithm. Our algorithm needs that robots have distincts identifiers, chirality, strong multiplicity detection (i.e. ability to count the number of colocated robots), memory (of size sublinear in the size of the ring and identifiers), and communication capacities but deals with (fully) anonymous ring. These assumptions (whose necessity is left as an open question here) are incomparable with those of Di Luna et al. [17] that assume anonymous but home-based robots (i.e. non fully anonymous rings). This algorithm brings two novelties with respect to the state-of-the-art: (*i*) it is the

first gracefully degrading algorithm dedicated to robot networks; and (*ii*) it is the first algorithm solving (a weak variant of) the gathering problem in the class $\mathcal{COT}$ (the largest class guaranteeing an exploitable recurrent property).

Roadmap. The organization of the paper follows. Section 2 presents formally the model we consider. Section 3 sums up impossibility results while Section 4 presents our gracefully degrading algorithm. Section 5 concludes the paper.

2 Model

Dynamic graphs. We consider the model of *evolving graphs* [19]. Time is discretized and mapped to $\mathbb{N}$. An evolving graph $\mathcal{G}$ is an ordered sequence $\{G_0, G_1, \ldots\}$ of subgraphs of a given static graph $G = (V, E)$ such that, for any $i \geq 0$, we call $G_i = (V, E_i)$ the snapshot of $\mathcal{G}$ at time i. Note that V is static and $|V|$ is denoted by n. We say that the edges of E_i are *present* in $\mathcal{G}$ at time i. G is the *footprint* of $\mathcal{G}$. The *underlying graph* of $\mathcal{G}$, denoted by $U_{\mathcal{G}}$, 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 such that there exists a time after which this edge is never present in $\mathcal{G}$. A *recurrent edge* is an edge of E that is not eventually missing. The *eventual underlying graph* of $\mathcal{G}$, denoted $U_{\mathcal{G}}^{\omega}$, is the static graph gathering all recurrent edges of $\mathcal{G}$ (i.e. $U_{\mathcal{G}}^{\omega} = (V, E_{\mathcal{G}}^{\omega})$ where $E_{\mathcal{G}}^{\omega}$ is the set of recurrent edges of $\mathcal{G}$). We only consider graphs whose footprints are anonymous and unoriented rings of size $n \geq 4$. The class $\mathcal{COT}$ (connected-over-time) contains all evolving graphs such that their eventual underlying graph is connected (note that there is at most one eventual missing edge in any ring of class $\mathcal{COT}$). The class $\mathcal{RE}$ (recurrent-edges) gathers all evolving graphs whose footprint contains only recurrent edges. The class $\mathcal{BRE}$ (bounded-recurrent-edges) includes all evolving graphs in which there exists a $\delta \in \mathbb{N}$ such that each edge of the footprint appears at least once every δ units of time. The class $\mathcal{AC}$ (always-connected) collects all evolving graphs where the graph G_i is connected for any $i \in \mathbb{N}$. The class $\mathcal{ST}$ (static) encompasses all evolving graphs where the graph G_i is the footprint for any $i \in \mathbb{N}$.

Robots. We consider systems of $\mathcal{R} \geq 4$ autonomous mobile entities called robots moving in a discrete and dynamic environment modeled by an evolving graph $\mathcal{G} = \{(V, E_0), (V, E_1) \ldots\}$, V being a set of nodes representing the set of locations where robots may be, E_i being the set of bidirectional edges through which robots may move from a location to another one at time i. Each robot knows n and $\mathcal{R}$. Each robot r possesses a distinct (positive) integer identifier id_r strictly greater than 0. Initially, a robot only knows the value of its own identifier. Robots have a persistent memory so they can store local variables.

Each robot r is endowed with strong local multiplicity detection, meaning that it is able to count the exact number of robots that are co-located with it at any time t. When this number equals 1, the robot r is *isolated* at time t. By opposition, we define a *tower* T as a couple (S, θ), where S is a set of robots with $|S| > 1$ 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 is maximal

for this property. We say that the robots of S form the tower at time t_s and that they are involved in the tower between time t_s and t_e. Robots are able to communicate (by direct reading) the values of their variables to each others only when they are involved in the same tower.

Finally, all the robots have the same 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 and all the robots agree on this labeling. Each robot r has a variable dir_r that stores the direction it currently *considers* (*right*, *left* or $\perp$).

Algorithms and execution. The *state* of a robot at time t corresponds to the values of its local variables at time t. The *configuration* γ_t of the system at time t gathers the snapshot at time t of the evolving graph, the positions (i.e. the nodes where the robots are currently located) and the state of each robot at time t. The *view* of a robot r at time t is composed of the state of r at time t, the state of all robots involved in the same tower as r at time t if any, and of the following local functions: $ExistsEdge(dir, round)$, with $dir \in \{right, left\}$ and $round \in \{current, previous\}$ which indicates if there exists an adjacent edge to the location of r at time t and $t-1$ respectively in the direction dir in G_t and in G_{t-1} respectively; $NodeMate()$ which gives the set of all the robots co-located with r (r is not included in this set); $NodeMateIds()$ which gives the set of all the identifiers of the robots co-located with r (excluded the one of r); and $HasMoved()$ which indicates if r has moved between time $t-1$ and t (see below).

The *algorithm* of a robot is written in the form of an ordered set of guarded rules $(label) :: guard \longrightarrow action$ where *label* is the name of the rule, *guard* is a predicate on the view of the robot, and *action* is a sequence of instructions modifying its state. Robots are uniform in the sense they share the same algorithm. Whenever a robot has at least one rule whose guard is true at time t, we say that this robot is *enabled* at time t. The local algorithm also specifies the initial value of each variable of the robot but cannot restrict its arbitrary initial position.

Given an evolving graph $\mathcal{G} = \{G_0, G_1, \ldots\}$ and an initial configuration γ_0, the *execution* σ in $\mathcal{G}$ starting from γ_0 of an algorithm is the maximal sequence $(\gamma_0, \gamma_1)(\gamma_1, \gamma_2)(\gamma_2, \gamma_3)\ldots$ 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 γ_i that is composed of three atomic and synchronous phases: Look, Compute, Move. During the Look phase, each robot captures its view at time i. During the Compute phase, each enabled robot executes the *action* associated to the first rule of the algorithm whose *guard* is true in its view. In the case the direction dir_r of a robot r is in $\{right, left\}$, the Move phase consists of moving r in the direction it considers if there exists an adjacent edge in that direction to its current node, otherwise (i.e. the adjacent edge is missing) r is *stuck* and hence remains on its current node. In the case where its direction is $\perp$, the robot remains on its current node.

3 Impossibility Results

This section presents a set of impossibility results (refer to Table 1) showing that some variants of the gathering problem cannot be solved depending on the dynamics of the ring in which the robots evolve and hence motivating our

gracefully degrading approach. First, we recall a result from Di Luna et al.. Note that differences between the considered models do not interfere with the proof.

Theorem 1 ([17]). *There exists no deterministic algorithm that satisfies $\mathbb{G}_E$ in rings of $\mathcal{AC}$ with size 4 or more for 4 robots or more.*

Note that Di Luna et al. provide only informal arguments for this impossibility result while we provide in the companion report [5] its full formal proof. It is possible to derive some other impossibility results from Theorem 1. Indeed, the inclusion $\mathcal{AC} \subset \mathcal{COT}$ allows us to state that $\mathbb{G}_E$ is also impossible under $\mathcal{COT}$.

Corollary 1. *There exists no deterministic algorithm that satisfies $\mathbb{G}_E$ in rings of $\mathcal{COT}$ with size 4 or more for 4 robots or more.*

From the very definitions of $\mathbb{G}$ and $\mathbb{G}_E$, it is straightforward to see that the impossibility of $\mathbb{G}_E$ under a given class implies the one of $\mathbb{G}$ under the same class.

Corollary 2. *There exists no deterministic algorithm that satisfies $\mathbb{G}$ in rings of $\mathcal{COT}$ or $\mathcal{AC}$ with size 4 or more for 4 robots or more.*

Finally, impossibility results for bounded variants of the gathering problem (i.e. the impossibility of $\mathbb{G}$ under $\mathcal{RE}$ and of $\mathbb{G}_W$ under $\mathcal{COT}$ and $\mathcal{RE}$) are obtained as follows. The definition of $\mathcal{COT}$ and $\mathcal{RE}$ does not exclude the ability for all edges of the graph to be missing initially and for any arbitrary long time, hence preventing the gathering of robots for any arbitrary long time if they are initially scattered. This observation is sufficient to prove a contradiction with the existence of an algorithm solving $\mathbb{G}$ or $\mathbb{G}_W$ in these classes.

Corollary 3. *There exists no deterministic algorithm that satisfies $\mathbb{G}$ or $\mathbb{G}_W$ in rings of $\mathcal{COT}$ or $\mathcal{RE}$ with size 4 or more for 4 robots or more.*

4 Gracefully Degrading Gathering

This section presents $\mathcal{GDG}$, our gracefully degrading gathering algorithm, that aims to solve different variants of the gathering under various dynamics (refer to Table 1). In the following, we informally describe our algorithm clarifying which variant of gathering is satisfied within which class of evolving graphs. Next, we present formally the algorithm and sketch its correctness proof.

Overwiew. Our algorithm has to overcome various difficulties. First, robots are evolving in an environment in which no node can be distinguished. So, the trivial algorithm in which the robots meet on a particular node is impossible. Moreover, since the footprint of the graph is a ring, (at most) one of the n edges may be an eventual missing edge. This is typically the case of classes $\mathcal{COT}$ and $\mathcal{AC}$. In that case, no robot is able to distinguish an eventual missing edge from a missing edge that will appear later in the execution. In particular, a robot stuck by a missing edge does not know whether it can wait for the missing edge to appear again or not. Finally, despite the fact that no robot is aware of which class of

dynamic graphs robots are evolving in, the algorithm is required to meet at least the specification of the gathering according to the class of dynamic graphs in which it is executed or a better specification than this one.

The overall scheme of the algorithm consists in first detecting r_{min}, the robot having the minimum identifier so that the $\mathcal{R}$ robots eventually gather on its node (i.e., satisfying specification $\mathbb{G}_E$). Of course, depending on the particular evolving graph in which our algorithm is executed, $\mathbb{G}_E$ may not achieved. In class $\mathcal{COT}$ and the "worst" possible evolving graph, one can expect specification $\mathbb{G}_{EW}$ only, i.e., at least $\mathcal{R}-1$ robots gathered.

The algorithm proceeds in four successive phases: `M` (for "am I the Min?"), `K` (for "min wait to be Known"), `W` (for "Walk"), and `T` (for "wait Termination"). Actually, again depending on the class of graphs and the evolving graph in which our algorithm is executed, we will see that the four phases are not necessarily all executed since the execution can be stopped prematurely, especially in case where $\mathbb{G}_E$ (or $\mathbb{G}$) is achieved. By contrast, they can also never be completed in some strong classes of dynamic graphs where the connectivity assumptions are weak (namely $\mathcal{AC}$ or $\mathcal{COT}$), solving $\mathbb{G}_{EW}$ (or $\mathbb{G}_W$) only.

Phase `M`. This phase leads each robot to know whether it possesses the minimum identifier. Initially every robot r considers the *right* direction. Then r moves to the *right* until it moves $4 * n * id_r$ steps on the right (where id_r is the identifier of r, and n is the size of the ring) or until it meets $\mathcal{R}-2$ other robots such that its identifier is not the smaller one among these robots or until it meets a robot that knows the identifier of r_{min}. The first robot that succeeds to move $4 * n * id_r$ steps in the right direction is necessarily r_{min}. Depending on the class of graph, one eventual missing edge may exist, preventing r_{min} to move on the *right* direction during $4 * n * id_{r_{min}}$ steps.

However, in the case where there is an eventual missing edge at least $\mathcal{R}-1$ robots succeed to be located on a same node. They are located either on the extremity of the eventual missing edge or on the extremity of a missing edge that is not eventually missing. The robot r_{min} is not necessarily located with these $\mathcal{R}-1$ robots gathered. Note that the weak form of gathering ($\mathbb{G}_{EW}$) could be solved in that case. However, the $\mathcal{R}-1$ robots gathered cannot stop their execution. Indeed, our algorithm aims at gathering the robots on the node occupied by r_{min}. However, r_{min} may not be part of the $\mathcal{R}-1$ robots that gathered. Further, it is possible for $\mathcal{R}-1$ robots to gather (without r_{min}) even when r_{min} succeeds in moving $4 * n * id_{r_{min}}$ steps to the right (i.e. even when r_{min} stops to move because it completed Phase `M`). In that case, if the $\mathcal{R}-1$ robots that gathered stop their execution, $\mathbb{G}_E$ cannot be solved in $\mathcal{RE}$, $\mathcal{BRE}$ and $\mathcal{ST}$ rings, as $\mathcal{GDG}$ should do. Note that, it is also possible for r_{min} to be part of the $\mathcal{R}-1$ robots that gathered.

Recall that robots can communicate when they are both located in the same node. So, the $\mathcal{R}-1$ robots may be aware of the identifier of the robot with the minimum identifier among them. Since it can or cannot be the actual r_{min}, let us call this robot $potentialMin$. Then, driven by $potentialMin$, a search phase starts during which the $\mathcal{R}-1$ robots try to visit all the nodes of the ring infinitely

Algorithm 1 Predicates used in $\mathcal{GDG}$

MinDiscovery() $\equiv [state_r = potentialMin \wedge \exists r' \in NodeMate(), (state_{r'} = righter \wedge id_r < id_{r'})] \vee [\exists r' \in NodeMate(), idMin_{r'} = id_r] \vee [\exists r' \in NodeMate(), (state_{r'} \in \{dumbSearcher, potentialMin\} \wedge id_r < idPotentialMin_{r'})] \vee [rightSteps_r = 4 * id_r * n]$

$\mathbb{G}_{\mathbf{E}}() \equiv |NodeMate()| = \mathcal{R} - 1$

$\mathbb{G}_{\mathbf{EW}}() \equiv |NodeMate()| = \mathcal{R} - 2 \wedge \exists r' \in \{r\} \cup NodeMate(), state_{r'} \in \{minWaitingWalker, minTailWalker\}$

HeadWalkerWithoutWalkerMate() $\equiv state_r = headWalker \wedge ExistsEdge(left, previous) \wedge \neg HasMoved() \wedge NodeMateIds() \neq walkerMate_r$

LeftWalker() $\equiv state_r = leftWalker$

HeadOrTailWalkerEndDiscovery() $\equiv state_r \in \{headWalker, tailWalker, minTailWalker\} \wedge walkSteps_r = n$

HeadOrTailWalker() $\equiv state_r \in \{headWalker, tailWalker, minTailWalker\}$

AllButTwoWaitingWalker() $\equiv |NodeMate()| = \mathcal{R} - 3 \wedge \forall r' \in \{r\} \cup NodeMate(), state_{r'} \in \{waitingWalker, minWaitingWalker\}$

WaitingWalker() $\equiv state_r \in \{waitingWalker, minWaitingWalker\}$

PotentialMinOrSearcherWithMinWaiting(r') $\equiv state_r \in \{potentialMin, dumbSearcher, awareSearcher\} \wedge state_{r'} = minWaitingWalker$

RighterWithMinWaiting(r') $\equiv state_r = righter \wedge state_{r'} = minWaitingWalker$

NotWalkerWithHeadWalker(r') $\equiv state_r \in \{righter, potentialMin, dumbSearcher, awareSearcher\} \wedge state_{r'} = headWalker$

NotWalkerWithTailWalker(r') $\equiv state_r \in \{righter, potentialMin, dumbSearcher, awareSearcher\} \wedge state_{r'} = minTailWalker$

PotentialMinWithAwareSearcher(r') $\equiv state_r = potentialMin \wedge state_{r'} = awareSearcher$

AllButOneRighter() $\equiv |NodeMate()| = \mathcal{R} - 2 \wedge \forall r' \in \{r\} \cup NodeMate(), state_{r'} = righter$

RighterWithSearcher(r') $\equiv state_r = righter \wedge state_{r'} \in \{dumbSearcher, awareSearcher\}$

PotentialMinOrRighter() $\equiv state_r \in \{potentialMin, righter\}$

DumbSearcherMinRevelation() $\equiv state_r = dumbSearcher \wedge \exists r' \in NodeMate(), (state_{r'} = righter \wedge id_{r'} > idPotentialMin_r)$

DumbSearcherWithAwareSearcher(r') $\equiv state_r = dumbSearcher \wedge state_{r'} = awareSearcher$

Searcher() $\equiv state_r \in \{dumbSearcher, awareSearcher\}$

often in both directions by subtle round trips. Doing so, r_{min} eventually knows that it possesses the actual minimum identifier.

Phase K. The goal of the second phase consists in spreading the identifier of r_{min} among the other robots. The basic idea is that during this phase, r_{min} stops moving and waits until $\mathcal{R} - 3$ other robots join it on its node so that its identifier is known by at least $\mathcal{R} - 3$ other robots. The obvious question arises: "*Why waiting for* $\mathcal{R} - 3$ *extra robots only?*". A basic idea to gather could be that once r_{min} is aware that it possesses the minimum identifier, it can just stop to move and just wait for the other robots to eventually reach its location, just by moving toward the right direction. Actually, depending on the particular

evolving graph considered one missing edge e may eventually appear, preventing robots from reaching r_{min} by moving toward the same direction only. That is why the gathering of the $\mathcal{R}-2$ robots is eventually achieved by the same search phase as in Phase `M` (since the search phase permits to at least 3 robots to explore infinitely often the nodes of the ring until reaching a given node). However, by doing this, it is possible to have 2 robots stuck on each extremity of e. Further, these two robots cannot change the directions they consider since a robot is not able to distinguish an eventual missing edge from a missing edge that will appear again later. This is why during Phase `K`, r_{min} stops to move until $\mathcal{R}-3$ other robots join it to form a tower of $\mathcal{R}-2$ robots. In this way these $\mathcal{R}-2$ robots start the third phase simultaneously.

Phase `W`. The third phase is a *walk* made by the tower of $\mathcal{R}-2$ robots. The $\mathcal{R}-2$ robots are split into two distinct groups, *Head* and *Tail*. Head is the unique robot with the maximum identifier of the tower. Tail, composed of $\mathcal{R}-3$ robots, is made of the other robots of the tower, led by r_{min}. Both move alternatively in the *right* direction during n steps such that between two movements of a given group the two groups are again located on a same node. This movement permits to prevent the two robots that do not belong to any of these two groups to be both stuck on different extremities of an eventual missing edge (if any) once this walk is finished. Since there exists at most one eventual missing edge, we are sure that if the robots that have executed the walk stop moving forever, then at least one robot can join them during the next and last phase.

As noted, it can exist an eventual missing edge, therefore, Head and Tail may not complete Phase `W`. Indeed, one of the two situations below may occur: (i) Head and Tail together form a tower of $\mathcal{R}-2$ robots but an eventual missing edge on their right prevents them to complete Phase `W`; (ii) Head and Tail are located on neighboring node and the edge between them is an eventual missing edge that prevents Head and Tail to continue to move alternatively.

Call u the node where Tail is stuck on an eventual missing edge. In the two situations described even if Phase `W` is not complete by both Head and Tail, either $\mathbb{G}_E$ or $\mathbb{G}_{EW}$ is solved. Indeed, in the first situation, necessarily at least one robot r succeeds to join u (either r considers the good direction to reach u or it meets a robot on the other extremity of the eventual missing edge that makes it change its direction, and hence makes it consider the good direction to reach u). In the second situation, necessarily at least two robots r and r' succeed to join u. This is done either because r and r' consider the good direction to reach u or because they reach the node where Head is located without Tail making them change their direction, and hence making them consider the good direction to reach u.

Once a tower of $\mathcal{R}-1$ robots including r_{min} is formed, $\mathbb{G}_{EW}$ is solved. Then, the latter robot tries to reach the tower to eventually solve $\mathbb{G}_E$ in favorable cases.

Phase `T`. The last phase starts once the robots of Head have completed Phase `W`. If it exists a time at which the robots of Tail complete Phase `W`, then Head and Tail form a tower of $\mathcal{R}-2$ robots and stop moving. As explained in the previous phase, Phase `W` ensures that at least one extra robot eventually joins the node where Head and Tail are located to form a tower of $\mathcal{R}-1$ robots. Once a tower

Algorithm 2 Functions used in $\mathcal{GDG}$

Function StopMoving()

$dir_r := \perp$

Function MoveLeft()

$dir_r := left$

Function BecomeLeftWalker()

$(state_r, dir_r) := (leftWalker, \perp)$

Function Walk()

$$dir_r := \begin{cases} \perp & \text{if } (id_r = idHeadWalker_r \wedge walkerMate_r \neq NodeMateIds()) \vee \\ & \quad (id_r \neq idHeadWalker_r \wedge idHeadWalker_r \in NodeMateIds()) \\ right & \text{otherwise} \end{cases}$$

$walkSteps_r := walkSteps_r + 1$ if $dir_r = right \wedge ExistsEdge(right, current)$

Function InitiateWalk()

$idHeadWalker_r := \text{MAX}(\{id_r\} \cup NodeMateIds())$

$walkerMate_r := NodeMateIds()$

$$state_r := \begin{cases} headWalker & \text{if } id_r = idHeadWalker_r \\ minTailWalker & \text{if } state_r = minWaitingWalker \\ tailWalker & \text{otherwise} \end{cases}$$

Function BecomeWaitingWalker(r')

$(state_r, idPotentialMin_r, idMin_r, dir_r) := (waitingWalker, id_{r'}, id_{r'}, \perp)$

Function BecomeMinWaitingWalker()

$(state_r, idPotentialMin_r, idMin_r, dir_r) := (minWaitingWalker, id_r, id_r, \perp)$

Function BecomeAwareSearcher(r')

$(state_r, dir_r) := (awareSearcher, right)$

$$(idPotentialMin_r, idMin_r) := \begin{cases} (idPotentialMin_{r'}, idPotentialMin_{r'}) \\ \qquad \text{if } state_{r'} = dumbSearcher \\ (idMin_{r'}, idMin_{r'}) \\ \qquad \text{otherwise} \end{cases}$$

Function BecomeTailWalker(r')

$(state_r, idPotentialMin_r, idMin_r) := (tailWalker, idPotentialMin_{r'}, idMin_{r'})$

$(idHeadWalker_r, walkerMate_r, walkSteps_r) :=$
$\quad (idHeadWalker_{r'}, walkerMate_{r'}, walkSteps_{r'})$

Function MoveRight()

$dir_r := right$

$rightSteps_r := rightSteps_r + 1$ if $ExistsEdge(dir, current)$

Function InitiateSearch()

$idPotentialMin_r := \text{MIN}(\{id_r\} \cup NodeMateIds())$

$$state_r := \begin{cases} potentialMin & \text{if } id_r = idPotentialMin_r \\ dumbSearcher & \text{otherwise} \end{cases}$$

$rightSteps_r := rightSteps_r + 1$ if $state_r = potentialMin \wedge ExistsEdge(dir, current)$

Function Search()

$$dir_r := \begin{cases} left & \text{if } |NodeMate()| \geq 1 \wedge id_r = \text{MAX}(\{id_r\} \cup NodeMateIds()) \\ right & \text{if } |NodeMate()| \geq 1 \wedge id_r \neq \text{MAX}(\{id_r\} \cup NodeMateIds()) \\ dir_r & \text{otherwise} \end{cases}$$

Algorithm 3 $\mathcal{GDG}$

Rules for Termination

$\mathbf{Term_1} :: \mathbb{G}_E() \longrightarrow$ **terminate**
$\mathbf{Term_2} :: \mathbb{G}_{EW}() \longrightarrow$ **terminate**

Rules for Phase `T`

$\mathbf{T_1} :: LeftWalker() \longrightarrow$ MoveLeft()
$\mathbf{T_2} :: HeadWalkerWithoutWalkerMate() \longrightarrow$ BecomeLeftWalker()
$\mathbf{T_3} :: HeadOrTailWalkerEndDiscovery() \longrightarrow$ StopMoving()

Rules for Phase `W`

$\mathbf{W_1} :: HeadOrTailWalker() \longrightarrow$ Walk()

Rules for Phase `K`

$\mathbf{K_1} :: AllButTwoWaitingWalker() \longrightarrow$ InitiateWalk()
$\mathbf{K_2} :: WaitingWalker() \longrightarrow$ StopMoving()
$\mathbf{K_3} :: \exists r' \in NodeMate(), PotentialMinOrSearcherWithMinWaiting(r')$
$\longrightarrow$ BecomeWaitingWalker(r')
$\mathbf{K_4} :: \exists r' \in NodeMate(), RighterWithMinWaiting(r') \wedge$
$ExistsEdge(right, current) \longrightarrow$ BecomeAwareSearcher(r')

Rules for Phase `M`

$\mathbf{M_1} :: PotentialMinOrRighter() \wedge MinDiscovery()$
$\longrightarrow$ BecomeMinWaitingWalker(r)
$\mathbf{M_2} :: \exists r' \in NodeMate(), NotWalkerWithHeadWalker(r') \wedge$
$ExistsEdge(right, current) \longrightarrow$ BecomeAwareSearcher(r')
$\mathbf{M_3} :: \exists r' \in NodeMate(), NotWalkerWithHeadWalker(r')$
$\longrightarrow$ BecomeAwareSearcher(r'); StopMoving()
$\mathbf{M_4} :: \exists r' \in NodeMate(), NotWalkerWithTailWalker(r')$
$\longrightarrow$ BecomeTailWalker(r'); Walk()
$\mathbf{M_5} :: \exists r' \in NodeMate(), PotentialMinWithAwareSearcher(r')$
$\longrightarrow$ BecomeAwareSearcher(r'); Search()
$\mathbf{M_6} :: AllButOneRighter() \longrightarrow$ InitiateSearch()
$\mathbf{M_7} :: \exists r' \in NodeMate(), RighterWithSearcher(r')$
$\longrightarrow$ BecomeAwareSearcher(r'); Search()
$\mathbf{M_8} :: PotentialMinOrRighter() \longrightarrow$ MoveRight()
$\mathbf{M_9} :: DumbSearcherMinRevelation() \longrightarrow$ BecomeAwareSearcher(r); Search()
$\mathbf{M_{10}} :: \exists r' \in NodeMate(), DumbSearcherWithAwareSearcher(r')$
$\longrightarrow$ BecomeAwareSearcher(r'); Search()
$\mathbf{M_{11}} :: Searcher() \longrightarrow$ Search()

of $\mathcal{R} - 1$ robots including r_{min} is formed, $\mathbb{G}_{EW}$ is solved. Then, the latter robot tries to reach the tower to eventually solve $\mathbb{G}_E$ in favorable cases. In the case the robots of Tail never complete the phase `W`, then this implies that Head and Tail are located on neighboring node and that the edge between them is an eventual missing edge. As described in Phase `W` either $\mathbb{G}_{EW}$ or $\mathbb{G}_E$ is solved.

Algorithm. Before presenting formally our algorithm, we first describe the set of variables of each robot. We recall that each robot r knows $\mathcal{R}$, n and id_r as

constants. In addition to the variable dir_r (initialized to *right*), each robot r possesses seven variables described below. Variable $state_r$ allows the robot r to know which phase of the algorithm it is performing and (partially) indicates which movement the robot has to execute. The possible values for this variable are *righter*, *dumbSearcher*, *awareSearcher*, *potentialMin*, *waitingWalker*, *minWaitingWalker*, *headWalker*, *tailWalker*, *minTailWalker* and *leftWalker*. Initially, $state_r$ is equal to *righter*. Initialized with 0, $rightSteps_r$ counts the number of steps done by r in the *right* direction when $state_r \in \{righter, potentialMin\}$. The next variable is $idPotentialMin_r$. Initially equals to -1, $idPotentialMin_r$ contains the identifier of the robot that possibly possesses the minimum identifier (a positive integer) of the system. This variable is especially set when $\mathcal{R}-1$ *righter* are located on a same node. In this case, the variable $idPotentialMin_r$ of each robot r that is involved in the tower of $\mathcal{R}-1$ robots is set to the value of the minimum identifier possessed by these robots. The variable $idMin_r$ indicates the identifier of the robot that possesses the actual minimum identifier among all the robots of the system. This variable is initially set to -1. Let $walkerMate_r$ be the set of all the identifiers of the $\mathcal{R}-2$ robots that initiate the Phase `W`. Initially this variable is set to $\emptyset$. The counter $walkSteps_r$, initially 0, maintains the number of steps done in the right direction while r performs the Phase `W`. Finally, the variable $idHeadWalker_r$ contains the identifier of the robot that plays the part of Head during the Phase `W`. Moreover, we assume the existence of a specific instruction: **terminate**. By executing this instruction, a robot stops executing the cycle Look-Compute-Move forever. To ease the writing of our algorithm, we define a set of predicates (presented in Algorithm 1) and functions (presented in Algorithm 2), that are used in our gracefully degrading algorithm $\mathcal{GDG}$. Recall that, during the Compute phase, only the first rule whose *guard* is true in the view of an enabled robot is executed.

Sketch of Proof. Due to the lack of space, in this section we only sketch the correctness proof of Algorithm $\mathcal{GDG}$. The interested reader may find the complete proofs in the companion report [5]. More precisely, we present which instance of the gathering our algorithm solves depending on the dynamics of the ring in which it is executed. In the following, we consider in the order the classes $\mathcal{COT}$, $\mathcal{AC}$, $\mathcal{RE}$, $\mathcal{BRE}$ and $\mathcal{ST}$. For ease of reading, we abuse the various values of the variable *state* to qualify the robots. For instance, if the current value of variable *state* of a robot is *righter*, then we say that the robot is a *righter* robot.

Theorem 2. *Algorithm $\mathcal{GDG}$ solves $\mathbb{G}_{EW}$ in $\mathcal{COT}$.*

Proof Outline. As the safety of $\mathbb{G}_{EW}$ directly follows from Rules $\mathbf{Term_1}$ and $\mathbf{Term_2}$, we only focus on its liveness in the following. The proof is done by analyzing successively each phase of $\mathcal{GDG}$.

In Phase `M`, r_{min} is supposed to be able, in finite time, to know that it possesses the minimum identifier among all the robots of the system. In our algorithm, a robot is aware that it possesses the minimum identifier when it is either a *minWaitingWalker* or a *minTailWalker* robot. Let us call *min* a robot such that its variable *state* is equal to one of these two values. To prove

the correctness of this phase, we prove first that only r_{min} can become *min* and then that r_{min} effectively becomes *min* in finite time.

First note that, by the rules of $\mathcal{GDG}$, if a robot is located on the same node as a *min*, it stops to be in Phase `M` and hence cannot be *min*. By the rules of $\mathcal{GDG}$, a robot is necessarily a $minWaitingWalker$ before becoming a $minTailWalker$. Moreover, only a *righter* or a $potentialMin$ can become a $minWaitingWalker$ (Rule $\mathbf{M_1}$). Therefore, if a robot becomes *min*, then necessarily it considers the right direction from the beginning of the execution until it becomes *min* (Rule $\mathbf{M_8}$). While executing the other phases of $\mathcal{GDG}$, a *min* can only consider either the $\perp$ or the right direction (refer to Rules $\mathbf{K_2}$, $\mathbf{K_1}$, $\mathbf{W_1}$, and $\mathbf{T_3}$). Besides, in the case a *min* robot r succeeds to execute all the phases of $\mathcal{GDG}$, it can only move from $4 * id_r * n + n$ steps in the right direction (refer to Rules $\mathbf{M_1}$, $\mathbf{K_2}$, $\mathbf{K_1}$, $\mathbf{W_1}$, and $\mathbf{T_3}$). Moreover, because of the dynamism of the ring, two robots r' and r'' such that both $state_{r'}$ and $state_{r''}$ belong to $\{righter, potentialMin\}$, can have their variables *rightSteps* such that $|rightSteps_{r'} - rightSteps_{r''}| \leq n$. Besides, it takes one round for a robot to update its variable *state* to *min*. Hence, since a *righter* or a $potentialMin$ can be located with a robot r just the round before r becomes *min*, this *righter* or $potentialMin$ can move again in the right direction during at most n steps without meeting the *min*. Hence, since for all $r \neq r_{min}$, $id_{r_{min}} < id_r$, we have $4 * id_{r_{min}} * n + n + n + n < 4 * id_r * n$. This implies that a robot r (with $r \neq r_{min}$) cannot become *min* thanks to the condition $rightSteps_r = 4 * id_r * n$ of the predicate $MinDiscovery()$ of Rule $\mathbf{M_1}$. Finally, the other conditions of the predicate $MinDiscovery()$ of Rule $\mathbf{M_1}$ cannot be satisfied by another robot than r_{min}. Indeed, by the rules of $\mathcal{GDG}$, a $potentialMin$ (resp. a *dumbSearcher*) is a robot that is aware of the identifiers of $\mathcal{R} - 1$ robots (Rule $\mathbf{M_6}$), and that possesses the minimum identifier among these $\mathcal{R} - 1$ robots (resp. and that keeps in its variable $idPotentialMin$ the value of the smallest identifier among these $\mathcal{R} - 1$ robots). Therefore, the first (resp. the third) condition of the predicate $MinDiscovery()$ of Rule $\mathbf{M_1}$ is true only for r_{min}. Finally, when there is no *min* in the execution, an *awareSearcher* (robot whose variable *idMin* is different from -1) is a robot that is aware of all the identifiers of all the robots of the system (Rules $\mathbf{M_5}$, $\mathbf{M_7}$, $\mathbf{M_9}$, and $\mathbf{M_{10}}$) and that keeps in its variable *idMin* the value of the minimum identifier among these robots. Thus, the second condition of the predicate $MinDiscovery()$ of Rule $\mathbf{M_1}$ is true only for r_{min}.

Then, we prove that r_{min} becomes *min* in finite time. First, note that as long as there is no *min* in the execution, r_{min} is either a *righter* or a $potentialMin$. In the case where r_{min} succeeds to move in the right direction during $4 * id_{r_{min}} * n$ steps, it becomes *min* (Rule $\mathbf{M_1}$). If r_{min} does not succeed to do so, then there exists an eventual missing edge, and necessarily $\mathcal{R} - 1$ *righter* succeed to be located on the same node. From this time, they are $potentialMin$ and *dumbSearcher* in the execution. It is also possible to have *awareSearcher* (Rules $\mathbf{M_5}$, $\mathbf{M_7}$, $\mathbf{M_9}$, and $\mathbf{M_{10}}$). As long as there is no *min*, *dumbSearcher* and *awareSearcher* execute the function SEARCH at each time (Rules $\mathbf{M_9}$, $\mathbf{M_{10}}$, and $\mathbf{M_{11}}$), and the $potentialMin$ executes either Rule $\mathbf{M_8}$ or function SEARCH (Rule $\mathbf{M_5}$).

By definition of SEARCH and of Rule $\mathbf{M_8}$, one robot succeeds to reach the node where r_{min} is stuck and to inform it that it has to become *min*.

Phase K is achieved when there are $\mathcal{R}-3$ $waitingWalker$ robots located on the same node as r_{min}, while r_{min} is a $minWaitingWalker$. By the rules of $\mathcal{GDG}$, as long as this phase is not achieved, there are only $righter$, $potentialMin$, $dumbSearcher$, $awareSearcher$, $waitingWalker$, and $minWaitingWalker$. By Rules $\mathbf{K_3}$ and $\mathbf{K_2}$, all the $waitingWalker$ and $minWaitingWalker$ are located on a same node and do not move. By analyzing the movements of the other kind of robots , we prove that it exists a time t at which this phase is achieved and that there is at most one $righter$ in the execution from time t.

Similarly, **Phase** W **and Phase** T are proved by analyzing the movements of the robots. At the time when the Phase K is achieved, we can prove that the two robots r_1 and r_2 that are not on the same node as the *min* are such that $state_{r_1} \in \{righter, potentialMin, awareSearcher, dumbSearcher\}$ and $state_{r_2} \in \{awareSearcher, dumbSearcher\}$. Moreover, once the Phase K is achieved, all the $waitingWalker$ and $minWaitingWalker$ execute Rule $\mathbf{K_1}$. While executing this rule the robot with the maximum identifier among these robots becomes $headWalker$, the $minWaitingWalker$ becomes $minTailWalker$ and the other robots become $tailWalker$. Analyzing all the possible movements of these kind of robots we succeed to prove that whatever the position of an eventual missing edge, in finite time, either Rule $\mathbf{Term_1}$ or Rule $\mathbf{Term_2}$ is executed. Hence, either $\mathcal{R}$ robots terminate their execution (Rule $\mathbf{Term_1}$) or $\mathcal{R}-1$ robots terminate their execution (Rule $\mathbf{Term_2}$) in finite time. □

Once Theorem 2 proved, classes inclusions and a careful analysis of the robot movements allow us to deduce the following set of results.

Theorem 3. *$\mathcal{GDG}$ solves $\mathbb{G}_W$ in $\mathcal{AC}$ in $O(id_{r_{min}} * n^2 + \mathcal{R} * n)$ rounds.*

Theorem 4. *$\mathcal{GDG}$ solves $\mathbb{G}_E$ in $\mathcal{RE}$.*

Theorem 5. *$\mathcal{GDG}$ solves $\mathbb{G}$ in $\mathcal{BRE}$ in $O(n * \delta * (id_{r_{min}} + \mathcal{R}))$ rounds.*

Corollary 4. *$\mathcal{GDG}$ solves $\mathbb{G}$ in $\mathcal{ST}$ in $O(n * (id_{r_{min}} + \mathcal{R}))$ rounds.*

5 Conclusion

In this paper, we apply for the first time the gracefully degrading approach to robot networks. This approach consists in circumventing impossibility results in highly dynamic systems by providing algorithms that adapt themselves to the dynamics of the graph: they solve the problem under weak dynamics and only guarantee that some weaker but related problems are satisfied whenever the dynamics increases and makes the original problem impossible to solve.

Focusing on the classical problem of gathering a squad of autonomous robots, we introduce a set of weaker variants of this problem that preserves its safety (in the spirit of the indulgent approach that shares the same underlying idea). Motivated by a set of impossibility results, we propose a gracefully degrading

gathering algorithm. We highlight that it is the first gracefully degrading algorithm dedicated to robot networks and the first algorithm focusing on the gathering in $\mathcal{COT}$, the class of dynamic graphs that exhibits the weakest recurrent connectivity.

A natural open question arises on the *optimality* of the graceful degradation we propose. Indeed, we prove that our algorithm provides for each class of dynamic graphs the best specification *among the ones we proposed*. We do not claim that another algorithm could not be able to satisfy stronger variants of the original gathering specification. Aside gathering in robot networks, defining a general form of *degradation optimality* seems to be a challenging future work.

References

1. D. Alistarh, S. Gilbert, R. Guerraoui, and C. Travers. Generating fast indulgent algorithms. *TCS*, 51(4):404–424, 2012.
2. M. Biely, P. Robinson, U. Schmid, M. Schwarz, and K. Winkler. Gracefully degrading consensus and k-set agreement in directed dynamic networks. *TCS*, 726:41–77, 2018.
3. M. Bournat, A. Datta, and S. Dubois. Self-stabilizing robots in highly dynamic environments. In *SSS*, pages 54–69, 2016.
4. M. Bournat, S. Dubois, and F. Petit. Computability of perpetual exploration in highly dynamic rings. In *ICDCS*, pages 794–804, 2017.
5. M. Bournat, S. Dubois, and F. Petit. Gracefully degrading gathering in dynamic rings. Technical report, arXiv 1805.05137, 2018.
6. N. Braud-Santoni, S. Dubois, M.-H. Kaaouachi, and F. Petit. The next 700 impossibility results in time-varying graphs. *IJNC*, 6(1):27–41, 2016.
7. A. Casteigts, P. Flocchini, B. Mans, and N. Santoro. Shortest, fastest, and foremost broadcast in dynamic networks. *IJFCS*, 26(4):499–522, 2015.
8. A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. *IJPEDS*, 27(5):387–408, 2012.
9. S. Dubois and R. Guerraoui. Introducing speculation in self-stabilization: an application to mutual exclusion. In *PODC*, pages 290–298, 2013.
10. P. Flocchini, E. Kranakis, D. Krizanc, N. Santoro, and C. Sawchuk. Multiple mobile agent rendezvous in a ring. In *LATIN*, pages 599–608, 2004.
11. R. Klasing, E. Markou, and A. Pelc. Gathering asynchronous oblivious mobile robots in a ring. In *ISAAC*, pages 744–753, 2006.
12. R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative byzantine fault tolerance. *TOCS*, 27(4):7:1–7:39, 2009.
13. F. Kuhn, N. Lynch, and R. Oshman. Distributed computation in dynamic networks. In *STOC*, pages 513–522, 2010.
14. L. Lamport. The part-time parliament. *TOCS*, 16(2):133–169, 1998.
15. M. Latapy, T. Viard, and C. Magnien. Stream graphs and link streams for the modeling of interactions over time. Technical report, arXiv 1710.04073, 2017.
16. G. Di Luna, S. Dobrev, P. Flocchini, and N. Santoro. Live exploration of dynamic rings. In *ICDCS*, pages 570–579, 2016.
17. G. Di Luna, P. Flocchini, L. Pagli, G. Prencipe, N. Santoro, and G. Viglietta. Gathering in dynamic rings. In *SIROCCO*, pages 339–355, 2017.
18. G. Di Stefano and A. Navarra. Optimal gathering of oblivious robots in anonymous graphs and its application on trees and rings. *DC*, 30(2):75–86, 2017.
19. B. Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *IJFCS*, 14(02):267–285, 2003.