Gracefully Degrading Gathering in Dynamic Rings

Marjorie Bournat, Swan Dubois, Franck Petit

Gracefully Degrading Gathering in Dynamic Rings

Marjorie Bournat, Swan Dubois and Franck Petit
Sorbonne Université, CNRS, Inria, LIP6, F-75005 Paris, France

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 that renders the original problem 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 should become 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.

1 Introduction

The classical approach in distributed computing consists in, first, fixing a set of assumptions that capture 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. In this spirit, many recent researches focus on defining pertinent assumptions for capturing the dynamics of those systems [7, 18, 23]. 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, 5, 6].

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. Then, a gracefully degrading algorithm guarantees to solve simultaneously the original problem under some assumption of dynamics and each of its variant 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 increases. 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, 9, 12] provide similar graceful degradation of the problem to satisfy with respect to synchrony –not with respect to dynamics. *Speculation* [8,13,17] 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 gracefully 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 the *gathering* in a *dynamic ring*. In this problem, starting from any initial position, robots must meet on an arbitrary location in a bounded time. 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 [7, 19, 23]. In this paper, we adopt the *evolving graph* model [23] 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.* [7]. The idea behind this hierarchy is to gather all dynamic graphs that share some temporal connectivity properties within classes. This allows to compare the strength of these temporal connectivity properties based on the inclusion of classes between them. We interest 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 do so recurrently; $\mathcal{BRE}$*–bounded-recurrent-edge* graphs– where any edge that appears at least once reappears

	$\mathbb{G}$	$\mathbb{G}_E$	$\mathbb{G}_W$	$\mathbb{G}_{EW}$
$\mathcal{COT}$	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. 6)	—	—	—

Table 1: Summary of our results. The symbol — means that a stronger variant of the problem is already proved solvable under the dynamics assumption. Our algorithm is gracefully degrading since it solves each variant of the gathering problem as soon as dynamics assumptions allow it.

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 the definition of these classes implies that $\mathcal{ST} \subset \mathcal{BRE} \subset \mathcal{RE} \subset \mathcal{COT}$ and $\mathcal{ST} \subset \mathcal{AC} \subset \mathcal{COT}$.

In robot networks, the gathering problem was extensively studied in the context of static graphs, *e.g.*, [10, 15, 16, 22]. 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 deals with the problem of exploration [3,4,11,14,20] –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 [21], 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 –all robots must gather in finite time on two adjacent nodes. They characterize the impact of chirality –ability of the robots to agree on a common orientation– and cross-detection –ability of the robots 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.* [21], in this paper we choose to 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 the liveness of the problem: at most one robot may not terminate or –not exclusively– all robots that terminate do so eventually. This choice is motivated by the approach adopted with indulgent algorithms [1, 9, 12]: 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 show then a set of impossibility results –summarized in Table 1– for these specifications for different classes of dynamic rings. Note that, in the case of $\mathbb{G}$ –the classical variant of the gathering problem–, our impossibility results in $\mathcal{COT}$ and $\mathcal{AC}$ subsume the one of Di Luna *et al.* [21] since we show that the result still holds even if robots are able to communicate, have identifiers, and not necessarily initially all scattered.

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 –refer to Table 1. Note that this challenging property is obtained without any knowledge nor detection of the dynamics by the robots that always execute the same algorithm.

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 of dynamic graphs that guarantees 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 proves the correctness of our gracefully degrading algorithm. Finally, Section 6 concludes the paper with some comments.

2 Model

In this section, we present a model borrowed from the one of [3] that proposes an extension to dynamic graphs of the classical model of robot networks in static graphs.

Dynamic graphs. In this paper, we consider the model of *evolving graphs* introduced in [23]. The 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 $\mathcal{G}$. In the following, we only consider graphs whose footprints are anonymous and unoriented rings of size $n \geq 4$. We define now formally the classes of dynamic graphs [7] we focus on. The class $\mathcal{COT}$ –connected-over-time graphs– contains all evolving graphs such that their eventual underlying graph is connected. The class $\mathcal{RE}$ –recurrent-edges graphs– gathers all evolving graphs whose footprint contains only recurrent edges. The class $\mathcal{BRE}$ –bounded-recurrent-edges graphs– 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 graphs– collects all evolving graphs where the graph G_i is connected for any $i \in \mathbb{N}$. The class $\mathcal{ST}$ –static graphs 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 representing connections 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. 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. We assume that each robot has a variable *dir* that stores the direction it currently *considers* –either *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 from the state of r at time t, the state of all robots involved in the same tower than 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 –the identifier of r is not included in this set; $HasMoved()$ which indicates if r has moved between time $t-1$ and t –see below.

The *algorithm* of a robot is written under the form of an ordered set of guarded rules $(label) :: guard \longrightarrow action$ where *label* is a name to refer to the rule in the text, *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 robot enabled by the algorithm executes the *action* associated to the first rule of the algorithm whose *guard* is true in this view. In the case the direction *dir* of a robot is in $\{right, left\}$, the Move phase consists of moving this robot 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– the robot is *stuck* and hence remains on its current node. In the case where the direction *dir* of a robot is $\perp$, the robot remains on its current node.

3 Impossibility Results

In this section, we present the set of impossibility results summarized in Table 1. These results show that some variants of the gathering cannot be solved depending on the dynamics of the ring in which the robots evolve and hence motivate our gracefully degrading approach.

First, we prove in Theorem 1 that $\mathbb{G}_E$ –the eventual variant of the gathering problem– is impossible to solve in $\mathcal{AC}$. Note that Di Luna *et al.* [21] provide a similar result but show it in an informal way only. Moreover, our result subsumes theirs since the considered models are different: we show that the result remains valid even if robots are identified, able to communicate, and not necessarily initially all scattered –other different assumptions exist between the two models but have no influence on our proof.

The proof of Theorem 1 relies on a generic framework introduced by Braud-Santoni *et al.* [5]. Note that, even if this generic framework is designed for another model –namely, the classical message passing model–, it is straightforward to borrow it for our current model. Indeed, as its proof only relies on the determinism of algorithms and indistinguishability of dynamic graphs, its arguments are directly translatable in our model. We present briefly this framework here. The interested reader is referred to the original work [5] for more details.

This framework is based on a result showing that, if we take a sequence of evolving graphs with ever-growing common prefixes –that hence converges to the evolving graph that shares all these common prefixes–, then the sequence of corresponding executions of any deterministic algorithm also converges. Moreover, we are able to describe the execution to which it converges as the execution of this algorithm in the evolving graph to which the sequence converges. This result is useful since it allows us to build counter-example in the context of impossibility results. Indeed, it is sufficient to construct an evolving graphs sequence –with ever-growing common prefixes– and to prove that their corresponding execution violates the specification of the problem for ever-growing time to exhibit an execution that never satisfies the specification of the problem.

Theorem 1. *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.*

Proof. By contradiction, assume that there exists a deterministic algorithm $\mathcal{A}$ that satisfies $\mathbb{G}_E$ in any ring of $\mathcal{AC}$ with size 4 or more for 4 robots or more. Let us choose arbitrarily two of these robots and denote them r_1 and r_2.

Note that $\mathcal{A}$ may allow the last robot to terminate only if it is co-located with all other robots (otherwise, we obtain a contradiction with the safety of $\mathbb{G}_E$). So, proving the existence of an execution of $\mathcal{A}$ in a ring of $\mathcal{AC}$ where r_1 and r_2 are never co-located is sufficient to obtain a contradiction with the liveness property of $\mathbb{G}_E$ and to show the result. This is the goal of the remainder of the proof.

To help the construction of this execution, we need introduce some notations as follows. Given an evolving graph $\mathcal{F}$, an edge $\tilde{e}$ of $\mathcal{F}$, and a time interval $\mathbb{I} \subseteq \mathbb{N}$, the evolving graph $\mathcal{F}\backslash\{\tilde{e}, \mathbb{I}\}$ is the evolving graph $\mathcal{F}'$ defined by: $e \in F'_i$ if and only if $e = \tilde{e} \wedge i \notin \mathbb{I} \wedge e \in F_i$ or $e \neq \tilde{e} \wedge e \in F_i$. Given an evolving graph $\mathcal{F}$ and two integers such that $t_1 \leq t_2$, we denote $\mathcal{F}^{t_1,\ldots,t_2}$ the subsequence $\{F_{t_1}, \ldots, F_{t_2}\}$ of $\mathcal{F}$. Given two evolving graphs, $\mathcal{F}$ and $\mathcal{H}$, and an integer t, the evolving graph $\mathcal{F}^{\{0,\ldots,t\}} \otimes \mathcal{H}^{\{t+1,\ldots,+\infty\}}$ is the evolving graph $\mathcal{F}'$ defined by: $e \in F'_i$ if and only if $i \leq t \wedge e \in F_i$ or $i > t \wedge e \in H_i$.

Let $\mathcal{G} = \{G_0, G_1, \ldots\}$ be a graph of $\mathcal{AC}$ whose footprint G is a ring of size 4 or more such that $\forall i \in \mathbb{N}, G_i = G$. Consider two nodes u and v of $\mathcal{G}$, such that the node v is the adjacent node of u in the footprint of $\mathcal{G}$ in the right direction. We denote by e_{uv} the edge linking the nodes u and v. Let $\mathcal{G}'$ be $\mathcal{G}\backslash\{e_{uv}, \mathbb{N}\}$. Let ε be the execution of $\mathcal{A}$ in $\mathcal{G}'$ starting from the configuration where r_1 is located on node u and r_2 is located on node v. Note that the distance in the footprint of $\mathcal{G}$ between r_1 and r_2 (denoted $d(r_1, r_2)$) is equal to one.

Our goal is to construct a sequence of rings of $\mathcal{AC}$ denoted $(\mathcal{G}_m)_{m\in\mathbb{N}}$ such that $\mathcal{G}_0 = \mathcal{G}'$ and, for any $i \geq 0$, r_1 and r_2 are never co-located before time t_i in ε_i (the execution of $\mathcal{A}$ in $\mathcal{G}_i$ starting from the same configuration as ε), $(t_m)_{m\in\mathbb{N}}$ being a strictly increasing sequence with $t_0 = 0$. First, we show in the next paragraph that, if some such $\mathcal{G}_i$ exists and moreover ensures the existence of a time $t'_i + 1 > t_i$ where the two robots are still on different nodes in ε_i, then we can construct $\mathcal{G}_{i+1}$. We prove, after that, that our construction guarantees the existence of such a t'_i, implying the well-definition of $(\mathcal{G}_m)_{m\in\mathbb{N}}$.

As r_1 and r_2 are not co-located at time t_i in ε_i, at least one of them must move in finite time in any execution starting from γ_{t_i} (otherwise, we obtain a contradiction with the liveness of $\mathbb{G}_E$). Let $t'_i \geq t_i$ be the smallest such time in the execution where the topology of the graph does not evolve from time t_i to time t'_i. In the following, we show how we construct the evolving graph $\mathcal{G}_{i+1}$, in function of t'_i and $\mathcal{G}_i$. As we assume that in $\mathcal{G}_i$, at time $t'_i + 1$, r_1 and r_2 are on two different nodes, *i.e.* $d(r_1, r_2) \geq 1$, the following cases are possible.

Case 1: $d(r_1, r_2) = 1$ at time $t'_i + 1$.
Denote e the edge between the respective locations of r_1 and r_2 at time $t'_i + 1$. We define $\mathcal{G}_{i+1}$ on the same footprint than $\mathcal{G}_i$ by $\mathcal{G}_{i+1} = \mathcal{G}_i^{0,\ldots,t'_i} \otimes (\mathcal{G}^{t'_i+1,\ldots,+\infty}\backslash\{e, \{t'_i + 1, \ldots, +\infty\}\})$.

Case 2: $d(r_1, r_2) = 2$ at time $t'_i + 1$.
Denote e and e' the two consecutive edges between the respective locations of r_1 and r_2 at time $t'_i + 1$.

We define first $\mathcal{G}'_i$ on the same footprint than $\mathcal{G}_i$ by $\mathcal{G}'_i = \mathcal{G}_i^{0,\dots,t'_i} \otimes \mathcal{G}^{t'_i+1,\dots,+\infty}$. Note that $\mathcal{G}'_i$ belongs to $\mathcal{AC}$ by assumption on $\mathcal{G}_i$ and since $\mathcal{G}$ is the static ring. Then, to avoid a contradiction with the liveness of $\mathbb{G}_E$, we know that there exists a time $\alpha_i \geq t'_i + 1$ in the execution of $\mathcal{A}$ on $\mathcal{G}'_i$ where at least one of our two robots move (*w.l.o.g.* assume that α_i is the smallest one). If, at time $\alpha_i + 1$, the two robots are on distinct nodes in $\mathcal{G}'_i$, then we define $\mathcal{G}_{i+1}$ on the same footprint than $\mathcal{G}_i$ by $\mathcal{G}_{i+1} = \mathcal{G}_i^{0,\dots,t'_i} \otimes \mathcal{G}^{t'_i+1,\dots,+\infty}$. If, at time $\alpha_i + 1$, the two robots are on a same node in $\mathcal{G}'_i$, then we define $\mathcal{G}_{i+1}$ on the same footprint than $\mathcal{G}_i$ by $\mathcal{G}_{i+1} = \mathcal{G}_i^{0,\dots,t'_i} \otimes (\mathcal{G}^{t'_i+1,\dots,+\infty} \backslash \{e, \{t'_i + 1, \dots, +\infty\}\})$.

Case 3: $d(r_1, r_2) > 2$ at time $t'_i + 1$.

We define $\mathcal{G}_{i+1}$ on the same footprint than $\mathcal{G}_i$ by $\mathcal{G}_{i+1} = \mathcal{G}_i^{0,\dots,t'_i} \otimes \mathcal{G}^{t'_i+1,\dots,+\infty}$.

Note that $\mathcal{G}_i$ and $\mathcal{G}_{i+1}$ are indistinguishable for robots until time t'_i. This implies that, at time $t'_i + 1$, r_1 and r_2 are on the same nodes in ε_i and in ε_{i+1}. By construction of t'_i, either r_1 or r_2 or both of the two robots move at time t'_i in ε_{i+1}. Moreover, by construction of $\mathcal{G}_i$, even if one or both of the robots move during the Move phase of time t'_i, at time $t'_i + 1$ the robots are still on two distinct nodes –since, in all cases above, either the distance between the robots before the move is strictly greater than 2, an edge between the two robots is missing before the move and prevents the meeting, or the two robots move in a way that prevents the meeting by indistinguishability between $\mathcal{G}_i$ and $\mathcal{G}_{i+1}$. Note that, by construction, $\mathcal{G}_{i+1}$ has at most one edge missing at each instant time and hence belongs to $\mathcal{AC}$.

Defining $t_{i+1} = t'_i + 1$, we succeed to construct $\mathcal{G}_{i+1}$ with the desired properties. Note that t'_i and $\mathcal{G}_0$ trivially satisfy all our assumptions. In other words, $(\mathcal{G}_m)_{m\in\mathbb{N}}$ is well-defined.

We can then define the evolving graph $\mathcal{G}_\omega$ such that $\mathcal{G}_\omega$ and $\mathcal{G}_0$ have the same footprint, and such that for all $i \in \mathbb{N}$, $\mathcal{G}_\omega$ shares a common prefix with $\mathcal{G}_i$ until time t'_i. As the sequence $(t_m)_{m\in\mathbb{N}}$ is increasing by construction, this implies that the sequence $(\mathcal{G}_m)_{m\in\mathbb{N}}$ converges to $\mathcal{G}_\omega$. Applying the theorem of Braud-Santoni *et al.* [5], we obtain that, until time t'_i, the execution of $\mathcal{A}$ in $\mathcal{G}_\omega$ is identical to the one in $\mathcal{G}_i$. This implies that, executing $\mathcal{A}$ in $\mathcal{G}_\omega$ (whose footprint is a ring of size 4 or more), r_1 and r_2 are always on distinct nodes, contradicting the liveness of $\mathbb{G}_E$ and proving the result. □

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 impossible under $\mathcal{COT}$ as well.

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 to 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

Along with Algorithms 1 and 2, Algorithm 3 (called $\mathcal{GDG}$) formally presents the program executed by each robot to gather. Being gracefully degrading, $\mathcal{GDG}$ is generic in the precise sense that it aims to solve different variants of the gathering under various dynamics—refer to Table 1. In Subsection 4.1, we informally describe the general scheme of our method, while at the same time clarifying cases in which $\mathcal{GDG}$ solves such or such variant of gathering within such or such class of evolving graphs. Next, Subsection 4.2 presents formally the algorithm.

4.1 Overwiew

In this subsection, we focus on the way that our method eventually gather either all or all but one robots. In other words, we omit to consider bounded termination issues, meaning that we consider only $\mathbb{G}_E$ –all robots

eventually gathered– and $\mathbb{G}_{EW}$ –all but one robots eventually gathered. Specifications $\mathbb{G}$ and $\mathbb{G}_W$ being related to the ability to bound the execution time are considered with $\mathbb{G}_E$ and $\mathbb{G}_{EW}$, respectively.

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 the 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 the same node as r_{min}—*i.e.*, satisfying Specification $\mathbb{G}_E$. Of course, depending on the class of dynamic graphs and the particular evolving graph in which our algorithm is executed, $\mathbb{G}_E$ may not achieved. In the weakest class (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`, `K`, `W`, and `T`. 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 weak settings (namely $\mathcal{AC}$ or $\mathcal{COT}$), solving $\mathbb{G}_{EW}$ (or $\mathbb{G}_W$) only.

Phase `M`. This phase leads each robot r to provide an answer to the question "*Am I the Min*?", *i.e.*, to know whether r possesses the minimum identifier among the $\mathcal{R}$ robots. To answer to this question, initially every robot r considers the *right* direction. Then r always move toward the *right* direction until it succeeds to move $4 * n * id_r$ steps on the right, where id_r is the identifier of r and n, the size of the ring. The first robot that succeeds to do so 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 that case at least $\mathcal{R}-1$ robots succeed to be located on a same node, but not necessarily the node where r_{min} is located. 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 to move during $4 * n * id_{r_{min}}$ right steps (*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 on a same node only. 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 often in both directions by subtile round trips. Doing so, by exchanging informations, r_{min} eventually knows that it possesses the actual minimum identifier among all the robots of the system.

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 class of graphs and the particular evolving graph in which our algorithm is executed, one missing edge e may eventually appear, preventing robots to reach 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`. 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 (among the $\mathcal{R}-2$ robots). Tail, composed of $\mathcal{R}-3$ robots, is made of the other robots of the tower, leaded by r_{min}. Both move

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

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 the footprint of the graph is a ring, there exists at most one eventual missing edge and 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 last and next 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.

1. Head and Tail together form a tower of $\mathcal{R} - 2$ robots but an eventual missing edge on their right prevent them to complete Phase `W`;

2. Head and Tail are located on neighboring node and the edge between them is an eventual missing edge that prevent 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. In fact, 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 considers 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 consider the good direction to reach u.

Once a tower of $\mathcal{R} - 1$ robots is formed, since r_{min} is among this tower, $\mathbb{G}_{EW}$ 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 of $\mathcal{R}-1$ robots is formed, since r_{min} is among this tower, $\mathbb{G}_{EW}$ 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 in this situation either $\mathbb{G}_{EW}$ or $\mathbb{G}_E$ is solved.

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

Function StopMoving()

$dir_r := \bot$

Function MoveLeft()

$dir_r := left$

Function BecomeLeftWalker()

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

Function Walk()

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

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

Function InitiateWalk()

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

$walkerMate_r := NodeMateIds()$

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

Function BecomeWaitingWalker(r')

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

Function BecomeMinWaitingWalker()

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

Function BecomeAwareSearcher(r')

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

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

Function BecomeTailWalker(r')

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

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

Function MoveRight()

$dir_r := right$

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

Function InitiateSearch()

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

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

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

Function Search()

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

4.2 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 $righter$, $dumbSearcher$, $awareSearcher$, $potentialMin$, $waitingWalker$, $minWaitingWalker$, $headWalker$, $tailWalker$,

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()

$minTailWalker$ 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 to execute 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.

5 Proofs of correctness of $\mathcal{GDG}$

In this section, we first prove, in Subsection 5.1, that $\mathcal{GDG}$ solves $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings. Then, in Subsection 5.2, we consider $\mathcal{AC}$, $\mathcal{RE}$, $\mathcal{BRE}$ and $\mathcal{ST}$ rings and for each of these classes of dynamic rings, we give the problem $\mathcal{GDG}$ solves in it.

We want to prove that, while executing $\mathcal{GDG}$, at least $\mathcal{R}-1$ robots terminate their execution on the same node. Therefore, in the proofs of correctness, we show that our algorithm forces the robots to execute either Rule $\mathbf{Term_1}$ or Rule $\mathbf{Term_2}$ whatever the harsh situation. Hence, the proofs are given in the case where these rules are not executed accidentally.

In the following, 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. Let us call min a robot such that its variable *state* is equal either to $minWaitingWalker$ or to

$minTailWalker$.

5.1 $\mathcal{GDG}$ solves $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings

In this subsection, we prove that $\mathcal{GDG}$ solves $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings. Since $\mathcal{GDG}$ is divided into four phases, we prove each of these phases hereafter.

5.1.1 Proofs of Correctness of Phase M

We recall that the goal of Phase M of our algorithm is to make the robot with the minimum identifier aware 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 *min*. Therefore, in this section we have to prove that only r_{min} can become *min*, and that r_{min} effectively becomes *min* in finite time. We prove this respectively in Lemmas 3 and 5.

First we give two observations that help us all along the proves of each phase.

Observation 1. *By the rules of $\mathcal{GDG}$, a robot whose state is not either righter or potentialMin cannot become a righter or a potentialMin.*

Observation 2. *By the rules of $\mathcal{GDG}$, a robot whose state is not righter cannot become a righter robot.*

While executing $\mathcal{GDG}$, once a robot knows that it possesses the minimum identifier, it remembers this information. In other words, once a robot becomes *min* it stays *min* during the rest of the execution. We prove this statement in the following lemma.

Lemma 1. *min is a closed state under $\mathcal{GDG}$.*

Proof. A robot is a *min* when its state is either equal to $minWaitingWalker$ or to $minTailWalker$. A $minTailWalker$ robot can only execute the rules $\mathbf{T_3}$ and $\mathbf{W_1}$ that do not update the variable *state*. A $minWaitingWalker$ robot can only execute the rules $\mathbf{K_1}$ and $\mathbf{K_2}$ that respectively makes it become a $minTailWalker$ and does not change its state. □

In the following lemma, we prove that *righter* and *potentialMin* are robots that always consider the right direction. This lemma helps us to prove the correctness of Phase M, as well as the correctness of Phase K.

Lemma 2. *If, at a time t, a robot is a righter or a potentialMin, then it considers the right direction from the beginning of the execution until the Look phase of time t.*

Proof. Robots that are *righter* robots in a configuration γ_i at time i and that are still *righter* in the configuration γ_{i+1}, consider the *right* direction during the move Phase of time i (Rule $\mathbf{M_8}$). Moreover, by Observation 2 and since initially all the robots are *righter* robots and consider the *right* direction, if a robot is a *righter* during the Look phase of a time t, this implies that it considers the *right* direction from the beginning of the execution until the Look phase of time t.

Similarly, robots that are *potentialMin* robots in a configuration γ_i at time i and that are still *potentialMin* in the configuration γ_{i+1}, consider the *right* direction during the move Phase of time i (Rule $\mathbf{M_8}$). The only way for a robot to become a *potentialMin* is to be a *righter* and to execute Rule $\mathbf{M_6}$. While executing Rule $\mathbf{M_6}$, a *righter* that becomes *potentialMin* does not change the direction it considers. Therefore, by Observations 1 and 2, and by the arguments of the first paragraph, this implies that if a robot is a *potentialMin* during the Look phase of a time t, then it considers the *right* direction from the beginning of the execution until the Look phase of time t. □

Now we prove one of the two main lemmas of this phase: we prove that only r_{min} can be aware that it possesses the minimum identifier among all the robots of the system.

Lemma 3. *Only r_{min} can become min.*

Proof. Assume that there exists a robot $r \neq r_{min}$ that becomes *min*. Assume also that r is the first robot different from r_{min} that becomes *min*. By definition of r_{min}, $id_r > id_{r_{min}}$.

A robot that is a *min* is a robot such that its variable *state* is either equal to $minWaitingWalker$ or to $minTailWalker$. A robot becomes $minTailWalker$ only if it executes Rule $\mathbf{K_1}$. A robot can execute Rule $\mathbf{K_1}$ only if it is a $minWaitingWalker$. A robot becomes $minWaitingWalker$ only if it executes Rule $\mathbf{M_1}$. Only *righter* robots or *potentialMin* robots can execute Rule $\mathbf{M_1}$ (refer to predicate $PotentialMinOrRighter()$). Then by Observation 1, we conclude that each robot can execute Rule $\mathbf{M_1}$ at most once. $(*)$

In the following, let us consider the different conditions of the predicate $MinDiscovery()$ of Rule $\mathbf{M_1}$ that permits r to become *min*.

Case 1: r becomes min because the condition "$\mathbf{state_r = potentialMin \wedge \exists r' \in NodeMate()}$, $\mathbf{(state_{r'} = righter \wedge id_r < id_{r'})}$" is true.

The only way for a robot to have its variable *state* set to $potentialMin$ is to execute Rule $\mathbf{M_6}$. This rule is executed when $\mathcal{R}-1$ *righter* robots are on a same node. Among these $\mathcal{R}-1$ *righter* robots, the one with the minimum identifier sets its variable *state* to $potentialMin$ while the other robots set their variables *state* to *dumbSearcher*. By Observation 1, a robot that becomes a *dumbSearcher* robot after the execution of Rule $\mathbf{M_6}$ can never become *righter* robot or $potentialMin$ robot. Moreover, by Observation 2, a robot that becomes a $potentialMin$ can never become a *righter*. Since $\mathcal{R}-1$ *righter* are needed to execute Rule $\mathbf{M_6}$, this rule can be executed only once during the execution. Therefore if r is a $potentialMin$, it is necessarily the robot that possesses the minimum identifier among the $\mathcal{R}-1$ robots that execute Rule $\mathbf{M_6}$. Moreover, if there exists a *righter* robot r' when r is $potentialMin$, this implies that r' has not executed Rule $\mathbf{M_6}$. Hence if $id_r < id_{r'}$, this necessarily implies that $r = r_{min}$, therefore there is a contradiction with the fact that $r \neq r_{min}$.

Case 2: r becomes min because the condition "$\mathbf{\exists r' \in NodeMate()}$, $\mathbf{idMin_{r'} = id_r}$" is true.

By $(*)$, r is not yet *min* at the time of its meeting with r'. A robot r' can update its variable $idMin$ with the identifier (other than its) of a robot that is not *min* only when it executes Rules $\mathbf{M_5}$, $\mathbf{M_7}$, $\mathbf{M_9}$ or $\mathbf{M_{10}}$. Among these rules only the rules $\mathbf{M_7}$ (in the case a *righter* is located with a *dumbSearcher*) and $\mathbf{M_9}$ permit a robot to update its variable $idMin$ with the identifier of a robot without copying the value of the variable $idMin$ of another robot. Therefore at least one of these rules is necessarily executed at a time, since initially the variables $idMin$ of the robots are equal to $\perp$. To execute Rule $\mathbf{M_7}$ (in the case a *righter* is located with a *dumbSearcher*) or Rule $\mathbf{M_9}$, a *dumbSearcher* robot must be present in the execution. Only the execution of Rule $\mathbf{M_6}$ permits to have *dumbSearcher* robots in the execution. This rule is executed when $\mathcal{R}-1$ *righter* robots are on a same node. The $\mathcal{R}-1$ robots that execute this rule, set their variables $idPotentialMin$ to the identifier of the robot that becomes $potentialMin$ while executing this rule. Moreover if a robot is a *dumbSearcher* in a configuration γ_t at time t and is still a *dumbSearcher* in the configuration γ_{t+1} then it does not update its variable $idPotentialMin$ during time t (since it executes Rule $\mathbf{M_{11}}$).

In the case Rule $\mathbf{M_7}$ is executed because a *righter* r_r is located with a *dumbSearcher* r_d necessarily $id_{r_r} > idPotentialMin_{r_d}$, otherwise it is not possible for r_r to execute Rule $\mathbf{M_7}$, since it would have executed Rule $\mathbf{M_1}$ at the same round (since the predicate $MinDiscovery()$ is true because $(state_{r_d} \in \{dumbSearcher, potentialMin\} \wedge id_{r_r} < idPotentialMin_{r_d})$). Therefore if Rule $\mathbf{M_7}$ is executed at round t because a *righter* r_r is located with a *dumbSearcher* r_d, this implies, by the predicate $DumbSearcherMinRevelation()$ of Rule $\mathbf{M_9}$, that Rule $\mathbf{M_9}$ is also executed at round t. Indeed, r_r executes Rule $\mathbf{M_7}$, while r_d executes Rule $\mathbf{M_9}$. The reverse is also true: if a *dumbSearcher* r_d executes Rule $\mathbf{M_9}$ at round t, then necessarily a *righter* r_r, such that $id_{r_r} > idPotentialMin_{r_d}$, executes Rule $\mathbf{M_7}$ at round t. While executing respectively these rules the two robots update their variables $idMin$ with the value of the variable $idPotentialMin$ of the *dumbSearcher*. By using the same arguments as the one used in case 1, we know that $idPotentialMin$ is the identifier of r_{min}. Therefore the variables $idMin$ are either set with the identifier of r_{min} while Rules $\mathbf{M_7}$ and $\mathbf{M_9}$ are executed, or copied from another robots while Rules $\mathbf{M_5}$ or $\mathbf{M_{10}}$ are executed. However whatever the rule executed the value of $idMin$ is set with the identifier of r_{min}.

Case 3: r becomes min because the condition "$\mathbf{\exists r' \in NodeMate()}$, $\mathbf{(state_{r'} \in \{dumbSearcher, potentialMin\} \wedge id_r < idPotentialMin_{r'})}$" is true.

Only the execution of Rule $\mathbf{M_6}$ permits to have *dumbSearcher* or $potentialMin$ in the execution. This rule is executed when $\mathcal{R}-1$ *righter* robots are on a same node. When executing this rule, the $\mathcal{R}-1$ robots set their variables $idPotentialMin$ to the identifier of the robot that possesses the minimum identifier among them. Moreover among the $\mathcal{R}-1$ robots that execute Rule $\mathbf{M_6}$, one robot becomes $potentialMin$ while the other become *dumbSearcher*. Besides if a robot is a *dumbSearcher* (resp. a $potentialMin$) in a configuration γ_t at time t and is still a *dumbSearcher* (resp. a $potentialMin$) in the configuration γ_{t+1} then it does not update its variable $idPotentialMin$ during time t since it executes Rule $\mathbf{M_{11}}$ (resp. $\mathbf{M_8}$). As Rule $\mathbf{M_6}$ can only be executed once (see the arguments of case 1), if r meets a *dumbSearcher* or a $potentialMin$ r', such that $id_r < idPotentialMin_{r'}$, this necessarily implies that r' is issued of the execution of Rule $\mathbf{M_6}$ while r has not executed this rule, and therefore $r = r_{min}$, which is a contradiction.

Case 4: r becomes min because $\mathbf{rightSteps_r = 4 * id_r * n}$.

At the time where r becomes *min*, r_{min} is either a *righter* robot, a $potentialMin$ robot or *min*, otherwise this implies that there already exists a *min* (other than r_{min}) in the execution, which is a contradiction with the fact that r is the first robot different from r_{min} that becomes *min*.

By the predicate $PotentialMinOrRighter()$ of Rule $\mathbf{M_1}$, only *righter* robots or $potentialMin$ robots can become *min*. By Lemma 2, if, at a time t, a robot is a *righter* or a $potentialMin$, then it considers the *right* direction from the beginning of the execution until the Look phase of time t. Robots that are *righter* robots or $potentialMin$ robots in a configuration γ_t at time t and that are either *righter* or $potentialMin$ in the configuration γ_{t+1} increase from 1 their variables *rightSteps* each time an adjacent edge in the right direction to their positions is present (Rules $\mathbf{M_6}$ and $\mathbf{M_8}$). Therefore, by the predicate $MinDiscovery()$ of Rule $\mathbf{M_1}$ a robot r" moves at most during $4 * id_{r"} * n$ steps in the right direction before being *min*.

By Lemma 1, from the time a robot becomes *min*, it is either a $minWaitingWalker$ or a $minTailWalker$. Therefore it can only execute Rules $\mathbf{Term_1}$, $\mathbf{Term_2}$, $\mathbf{K_1}$, $\mathbf{K_2}$, $\mathbf{W_1}$ and $\mathbf{T_3}$. This implies that once a robot is *min*, it considers only either the *right* or the $\perp$ direction, and can move during at most n steps in the right direction before stopping to move definitively (by executing the following rules in the order: $\mathbf{K_2}$, $\mathbf{K_1}$, $\mathbf{W_1}$ and $\mathbf{T_3}$). Therefore by the previous paragraph, a *min* r" considers the right or the $\perp$ direction from the beginning of the execution until the end of the execution, and can move during at most $4 * id_{r"} * n + n$ steps in the right direction during the whole execution.

Because of the dynamism of the ring, by Observation 1 and since when a *righter* or a $potentialMin$ robot stops to be a *righter* or a $potentialMin$ robot, it stops to update the value of its variable *rightSteps*, we have: $\forall r_1, r_2 \in \mathcal{R}^2, state_{r_1}, state_{r_2} \in \{righter, potentialMin\}^2, |rightSteps_{r_1} - rightSteps_{r_2}| \leq n$.

Because it takes one round for a robot to update its variable *state* to *min*, a *righter* or a $potentialMin$ can be located with a robot r just the round before r becomes *min*. Therefore this *righter* or $potentialMin$ can move again in the right direction during at most n steps without meeting the *min*.

We know that $id_{r_{min}} < id_r$, therefore we have $4 * id_{r_{min}} * n + n + n + n < 4 * id_r * n$. Hence there exists a time at which r meets r_{min} while r_{min} is *min* and r is not yet *min*. At this time, by the rules of $\mathcal{GDG}$, r stops being a *righter* or a $potentialMin$ robot, and hence by Observation 1, r cannot be anymore a *righter* robot or a $potentialMin$ robot and therefore it cannot become *min*, which leads to a contradiction.

□

The following lemma helps us to prove the Lemma 5. This lemma is true only if there is no *min* in the execution. In other words, it is true only if all the robots are executing Phase `M`.

Lemma 4. *If there is no min in the execution, if, at time t, a robot r is such that $state_r \in \{dumbSearcher, awareSearcher\}$, then, during the Move phase of time $t-1$, it does not consider the $\perp$ direction.*

Proof. Consider a robot r such that, at time t, $state_r \in \{dumbSearcher, awareSearcher\}$.

While executing $\mathcal{GDG}$, since initially all the robots are *righter*, if there is no *min*, only *righter*, $potentialMin$, *dumbSearcher* and *awareSearcher* robots can be present in the execution.

Consider then the two following cases.

Case 1: At time $t-1$, r is neither a *dumbSearcher* nor an *awareSearcher*.

Whatever the state of r at time $t-1$ (*righter* or $potentialMin$), to have its variable state at time t equals either to *dumbSearcher* or to *awareSearcher*, r executes at time $t-1$ either Rule $\mathbf{M_5}$, $\mathbf{M_6}$ or $\mathbf{M_7}$.

Consider first the case where r executes Rule $\mathbf{M_6}$ at time $t-1$. Only *righter* robots can execute Rule $\mathbf{M_6}$. While executing Rule $\mathbf{M_6}$, r becomes a *dumbSearcher* (since while executing this rule a *righter* can become either a *dumbSearcher* or a $potentialMin$). Moreover, while executing Rule $\mathbf{M_6}$, a *righter* that becomes *dumbSearcher* does not change the direction it considers. By Lemma 2, during the Look phase of time $t-1$, r considers the right direction and since r does not change its direction during the Compute phase of time $t-1$, this implies that the lemma is proved in this case.

Consider now the case where r executes either Rule $\mathbf{M_5}$ or $\mathbf{M_7}$. While executing these rules the function Search is called.

While executing the function Search, if there are multiple robots on the current node of r at time $t-1$, it considers either the right or the left direction. Therefore, in this case the lemma is proved.

In the case r is alone on its node at time $t-1$, while executing the function Search it does not change its direction. Moreover, while executing Rules $\mathbf{M_5}$ or $\mathbf{M_7}$, before calling the function Search the robot calls the function BecomeAwareSearcher that sets its direction to the right direction. Therefore, in these cases, even if r is alone on its node, it considers a direction different from $\perp$ during the Move phase of time $t-1$, hence the lemma is proved.

Case 2: At time t − 1, r is a *dumbSearcher* or an *awareSearcher*.

Whatever the state of r at time $t-1$ (*dumbSearcher* or *awareSearcher*), to have its variable state at time t equals either to *dumbSearcher* or to *awareSearcher*, r executes at time $t-1$ either Rule $\mathbf{M_9}$, $\mathbf{M_{10}}$ or $\mathbf{M_{11}}$. While executing these rules the function SEARCH is called.

As highlighted in the case 1, if there are multiple robots on the current node of r at time $t-1$, the lemma is proved.

Moreover, while executing Rules $\mathbf{M_9}$ and $\mathbf{M_{10}}$, before calling the function SEARCH the robot calls the function BECOMEAWARESEARCHER that sets its direction to the right direction. Therefore, in these cases, even if r is alone on its node, it considers a direction different from $\perp$ during the Move phase of time $t-1$, hence the lemma is proved.

It remains the case where r executes Rule $\mathbf{M_{11}}$ at time $t-1$ while it is alone on its node. In this case, while executing Rule $\mathbf{M_{11}}$, r does not change its direction (refer to the function SEARCH). Since at time $t-1$, r is already a *dumbSearcher* or an *awareSearcher*, and since initially all the robots are *righter*, by recurrence on all the cases treated previously (Case 1 and 2), the direction r considers during the Move phase of time $t-1$ cannot be equal to $\perp$.

□

Finally, we prove the other main lemma of this phase: we prove that r_{min} is aware, in finite time, that it possesses the minimum identifier among all the robots of the system.

Lemma 5. *In finite time r_{min} becomes min.*

Proof. Assume that r_{min} does not become *min*. By Lemma 3, only r_{min} can be *min*. While executing $\mathcal{GDG}$, since initially all the robots are *righter*, if there is no *min*, only *righter*, *potentialMin*, *dumbSearcher* and *awareSearcher* robots can be present in the execution.

Initially all the robots are *righter*. In the case where there is no *min* in the execution, by the rules of $\mathcal{GDG}$, from a configuration γ_t at a time t where there are only *righter* robots, it is not possible to have *awareSearcher* in the configuration γ_{t+1}. A robot can become a *dumbSearcher* or a *potentialMin* only when Rule $\mathbf{M_6}$ is executed. This rule is executed when $\mathcal{R}-1$ *righter* robots are on a same node (refer to predicate *AllButOneRighter*()).

Let us now consider the three following cases that can occur in the execution.

Case 1: Rule $\mathbf{M_6}$ is never executed.

In this case all the robots are *righter* robots during the whole execution, and execute therefore Rule $\mathbf{M_8}$ at each instant time. While executing Rule $\mathbf{M_8}$, a robot always considers the *right* direction and increments its variable *rightSteps* by one each time there exists an adjacent *right* edge to its location. Since by assumption r_{min} does not become *min*, then by Rule $\mathbf{M_1}$ and predicate *MinDiscovery*(), r_{min} cannot succeed to have its variable *rightSteps* equals to $4 * id_{r_{min}} * n$, otherwise the lemma is true. Therefore it exists a time at which r_{min} is on a node such that its adjacent *right* edge is missing forever. Since it can exist at most one eventual missing edge in a $\mathcal{COT}$ ring, and since all the robots always move in the *right* direction when there is an adjacent right edge to their location (since they execute Rule $\mathbf{M_8}$), it exists a time at which $\mathcal{R}-1$ *righter* robots are on a same node, cases 2 and 3 are then considered.

Case 2: Rule $\mathbf{M_6}$ is executed but $\mathbf{r_{min}}$ is not among the $\mathcal{R}-1$ righter robots that execute it.

While executing Rule $\mathbf{M_6}$, among the $\mathcal{R}-1$ *righter* located on a same node that execute this rule, the robot with the minimum identifier r_p becomes *potentialMin* while the other robots become *dumbSearcher*, and all update their variables *idPotentialMin* to id_{r_p}. By definition we have $id_{r_p} > id_{r_{min}}$. By Observation 1, a robot that becomes a *dumbSearcher* can never become *righter* robot or *potentialMin* robot. Moreover, by Observation 2, a robot that becomes a *potentialMin* can never become a *righter*. Since $\mathcal{R}-1$ *righter* are needed to execute Rule $\mathbf{M_6}$, this rule can be executed only once. Note that if a robot is a *dumbSearcher* (resp. a *potentialMin*) in a configuration γ_t at time t and is still a *dumbSearcher* (resp. a *potentialMin*) in the configuration γ_{t+1} then it does not update its variable *idPotentialMin* during time t since it executes Rule $\mathbf{M_{11}}$ (resp. $\mathbf{M_8}$)

At the time of the execution of Rule $\mathbf{M_6}$, r_{min} is a *righter*, since it is not among the robots that execute this rule. After the execution of this rule r_{min}, as a *righter*, cannot meet a *potentialMin* robot. Indeed the only way for a robot to become *potentialMin* is to execute Rule $\mathbf{M_6}$. Therefore only r_p can be *potentialMin*, and we know that $idPotentialMin_{r_p} = id_{r_p} > id_{r_{min}}$. Hence if r_{min} meets a *potentialMin*, then by Rule $\mathbf{M_1}$ and predicate *MinDiscovery*() the lemma is true, which is a contradiction.

Similarly, r_{min} as a *righter* cannot meet a *dumbSearcher* r_d. Indeed, only Rule $\mathbf{M_6}$ permits a robot to become a *dumbSearcher*. Therefore, since $idPotenialMin_{r_d} = id_{r_p} > id_{r_{min}}$, if r_{min} meets a *dumbSearcher*, then by Rule $\mathbf{M_1}$ and predicate $MinDiscovery()$ the lemma is true, which is a contradiction.

Moreover it cannot exist *awareSearcher* in this execution. Indeed, as said previously, from a configuration γ_t at a time t where there are only *righter* robots, it is not possible to have *awareSearcher* in the configuration γ_{t+1}. Therefore *awareSearcher* can be present in the execution only after the execution of Rule $\mathbf{M_6}$. In the case where there is not yet *awareSearcher*, a robot can become an *awareSearcher* only if a *righter* meets a *dumbSearcher* (Rules $\mathbf{M_9}$ and $\mathbf{M_7}$). However after the execution of Rule $\mathbf{M_6}$, only r_{min} is a *righter*, and as explained in the previous paragraph, if r_{min} as a *righter* meets a *dumbSearcher* there is a contradiction.

Since there is no *awareSearcher* and since r_{min} as a *righter* cannot meet neither *potentialMin* nor *dumbSearcher*, this implies that r_{min} stays a *righter* during the whole execution and therefore executes Rule $\mathbf{M_8}$ at each instant time. By the same arguments as the one used in case 1, necessarily it exists a time at which r_{min} is on node such that its adjacent *right* edge is missing forever, otherwise the lemma is true. However since there is no *min* is the execution, and there is no *awareSearcher*, r_p stays a *potentialMin* and executes Rule $\mathbf{M_8}$ at each instant time, therefore it always considers the *right* direction. Since it can only exist one eventual missing edge and since this edge is the adjacent *right* edge to the position of r_{min}, all the other edges are infinitely often present. Therefore, in finite time, the *potentialMin* is located on the same node as r_{min}, which is a contradiction.

Case 3: Rule $\mathbf{M_6}$ is executed and $\mathbf{r_{min}}$ is among the $\mathcal{R}-1$ righter robots that execute it.

We use the same arguments as the one used in case 2. Therefore we know that while executing Rule $\mathbf{M_6}$, r_{min} becomes *potentialMin*, since r_{min} possesses the minimum identifier among all the robots of the system.

Moreover, since r_{min} does not become *min*, as a *potentialMin*, it cannot meet a *righter* robot otherwise by Rule $\mathbf{M_1}$ and predicate $MinDiscovery()$ the lemma is true.

Similarly, r_{min} as a *potentialMin* cannot meet *awareSearcher*. Indeed in the case there is not yet *awareSearcher*, a robot can become an *awareSearcher* only if a *righter* meets a *dumbSearcher* (Rules $\mathbf{M_9}$ and $\mathbf{M_7}$). While executing these rules a robot that becomes an *awareSearcher* sets its variable *idMin* to the identifier of the variable *potentialMin* of the *dumbSearcher*, which is in this case $id_{r_{min}}$. An *awareSearcher* never updates the value of its variable *idMin*. Once there is at least one *awareSearcher* in the execution, it is possible to have other robots that become *awareSearcher* thanks to the execution of Rule $\mathbf{M_{10}}$. However while executing this rule, a robot that becomes *awareSearcher* copies the value of the variable *idMin* of the *awareSearcher* it is located with. Therefore if r_{min}, as a *potentialMin*, meets an *awareSearcher*, by Rule $\mathbf{M_1}$ and predicate $MinDiscovery()$, the lemma is true, which is a contradiction.

Therefore, as a *potentialMin*, r_{min} executes Rule $\mathbf{M_8}$ at each instant time. By the same arguments as the one used in case 1, necessarily it exists a time at which r_{min} is on node such that its adjacent *right* edge is missing forever, otherwise the lemma is true.

By Observation 1, *dumbSearcher* and *awareSearcher* robots cannot become *righter* or *potentialMin*. As explained, if there is no meeting between a *dumbSearcher* robot and a *righter* robot, it cannot exist *awareSearcher* robots in the execution. As seen previously, no *righter* robot can meet r_{min}. At the time where Rule $\mathbf{M_6}$ is executed there is a *righter* robot r in the execution. In the case r never meets a *dumbSearcher* robot, it executes Rule $\mathbf{M_8}$ at each instant time. Hence, using the arguments as the one used in case 2, in finite time, r can be located on the same node as r_{min}, which is a contradiction. This implies that there exists a time at which r, as a *righter* robot, meets at least a *dumbSearcher* robot r'. In this case r executes Rule $\mathbf{M_7}$ (refer to the predicate $RighterWithSearcher()$) and all the *dumbSearcher* robots located with r including r' execute Rule $\mathbf{M_9}$ (by the predicate $DumbSearcherMinRevelation()$ and since $id_r > id_{r_{min}}$). Hence r and all the *dumbSearcher* robots located with r become *awareSearcher* robots and execute the function SEARCH. When a robot executes the function SEARCH while there are multiple robots on its node, if it possesses the maximum identifier among the robots of its node, it considers the left direction, otherwise it considers the right direction. Therefore, once $\mathbf{M_7}$ and $\mathbf{M_9}$ are executed, there are at least two *awareSearcher* considering two opposite directions. Moreover once $\mathbf{M_7}$ and $\mathbf{M_9}$ are executed, except r_{min} there are only *dumbSearcher* and *awareSearcher* robots in the execution. When a *dumbSearcher* robot meets an *awareSearcher* robot, it executes Rule $\mathbf{M_{10}}$ and therefore becomes *awareSearcher* robot and executes the function SEARCH. An *awareSearcher* executes Rule $\mathbf{M_{11}}$ at each instant time, therefore it calls the function SEARCH at each instant time. While executing the function SEARCH, if an *awareSearcher* robot is alone on its node, it considers the last direction it considers (this direction cannot be equal to $\perp$ by Lemma 4). All this implies that in finite time an *awareSearcher* robot

is located on the same node as r_{min}. Therefore by Rule $\mathbf{M_1}$ and predicate $MinDiscovery()$, r_{min} becomes min.

□

By Lemmas 3 and 5, we can deduce the following corollary which proves the correctness of Phase M.

Corollary 4. *Only r_{min} becomes min in finite time.*

5.1.2 Proofs of Correctness of Phase K

Once r_{min} completes Phase M, it stops to move and waits for the completion of Phase K. We recall that, during Phase K of $\mathcal{GDG}$, $\mathcal{R}-3$ robots must join r_{min} on the node where it is waiting. More precisely, while executing $\mathcal{GDG}$, Phase K is achieved when $\mathcal{R}-3$ $waitingWalker$ robots are located on the node where r_{min}, as min, is waiting. In the previous subsection, we prove that, in finite time, only r_{min} becomes min (Corollary 4) and that once r_{min} is min it stays min for the rest of the execution (Lemma 1). Note that, by the rules of $\mathcal{GDG}$, the min is necessarily a $minWaitingWalker$ robot before being a $minTailWalker$ (since only a $minWaitingWalker$ can become a $minTailWalker$ while executing Rule $\mathbf{K_1}$). Moreover, by Rule $\mathbf{K_2}$, r_{min}, as a $minWaitingWalker$, does not move until $\mathcal{R}-3$ $waitingWalker$ robots are on its node. Therefore, as $minWaitingWalker$, r_{min} is, as expected, always on the same node. Let u be the node on which r_{min}, as a $minWaitingWalker$, is located. Let t_{min} be the time at which r_{min} becomes a $minWaitingWalker$ robot. In this subsection, we consider the execution from time t_{min}.

To simplify the proofs, we introduce the notion of $towerMin$ as follows.

Definition 1 ($towerMin$)**.** *A $towerMin$ corresponds to a configuration of the execution in which $\mathcal{R}-3$ $waitingWalker$ robots are located on the same node as the $minWaitingWalker$.*

To prove the correctness of Phase K, we hence have to prove that, in finite time, a $towerMin$ is formed.

As noted previously, by the rules of $\mathcal{GDG}$, as long as there is no $towerMin$, r_{min} stays a $minWaitingWalker$ robot.

The following observation is useful to prove the correctness of this phase.

Observation 3. *There exists no rule in $\mathcal{GDG}$ permitting a robot that stops being either $minWaitingWalker$ or $waitingWalker$ robot to be again a $minWaitingWalker$ or $waitingWalker$ robot.*

To prove the correctness of this phase, we prove, first, that if a $potentialMin$ is present in the execution then, in finite time, a $towerMin$ is present in the execution, next, we prove that if there is no $potentialMin$ in the execution then, in finite time, a $towerMin$ is also present in the execution. We prove this respectively in Lemmas 15 and 16. To simplify the proofs of these two lemmas, we need to prove the nine following lemmas.

In the following lemma we prove that it can exist at most one $towerMin$ in the whole execution.

Lemma 6. *It can exist at most one $towerMin$ in the whole execution.*

Proof. By definition a $towerMin$ is composed of one $minWaitingWalker$ and $\mathcal{R}-3$ $waitingWalker$ robots. Once a $towerMin$ is formed, the $\mathcal{R}-2$ ($\mathcal{R}-2 \geq 2$) robots involved in the $towerMin$ execute Rule $\mathbf{K_1}$. While executing this rule the robot with the maximum identifier among the $\mathcal{R}-2$ robots involved in the $towerMin$ becomes $headWalker$ while the $minWaitingWalker$ becomes $minTailWalker$ and the other robots involved in the $towerMin$ become $tailWalker$.

Then by Observation 3 and since by Corollary 4 only r_{min} can be $minWaitingWalker$, the lemma is proved. □

In the following lemma, we prove that all the $waitingWalker$ as well as the $minWaitingWalker$ are located on node u and do not move. This is important to prove that, in finite time, a $towerMin$ is formed.

Lemma 7. *All $waitingWalker$ robots are located on the same node as r_{min} when $state_{r_{min}} = minWaitingWalker$ and neither the $waitingWalker$ robots nor r_{min}, as a $minWaitingWalker$, move.*

Proof. By the rules of $\mathcal{GDG}$, as long as there is no $towerMin$, r_{min} is $minWaitingWalker$. While r_{min} is the $minWaitingWalker$, it executes Rule $\mathbf{K_2}$ at each instant time. While executing this rule, r_{min} considers the $\perp$ direction and therefore does not move.

Only Rule $\mathbf{K_3}$ permits a robot r to become a $waitingWalker$ robot. For this rule to be executed r must be located with a $minWaitingWalker$ (refer to predicate $PotentialMinOrSearcherWithMin()$). By Corollary 4, only r_{min} can be $minWaitingWalker$. While executing Rule $\mathbf{K_3}$, r considers the $\perp$ direction and therefore at the time of the execution of this rule, r does not move and is on the node where r_{min}, as a $minWaitingWalker$, is located.

While r is a $waitingWalker$ robot, as long as there is no $towerMin$ in the execution, it executes Rule $\mathbf{K_2}$ at each instant time. Therefore r does not move. As noted previously, the location where r stops moving is the location where r_{min}, as the $minWaitingWalker$, is located.

Once a $towerMin$ is present in the execution the $waitingWalker$ robots and the $minWaitingWalker$ composing this $towerMin$ execute Rule $\mathbf{K_1}$. While executing this rule the robots do not change the direction they consider and stop being $waitingWalker/minWaitingWalker$ robots. Therefore, by Observation 3 and since by Corollary 4 only r_{min} can be $minWaitingWalker$, all $waitingWalker$ robots are located on the same node as r_{min} when $state_{r_{min}} = minWaitingWalker$ and neither the $waitingWalker$ robots nor r_{min}, as a $minWaitingWalker$, move. □

Now we prove a property on $potentialMin$.

Lemma 8. *It can exist at most one potentialMin robot in the whole execution.*

Proof. Only the execution of Rule $\mathbf{M_6}$ permits a robot to become a $potentialMin$ robot. Rule $\mathbf{M_6}$ is executed when $\mathcal{R}-1$ $righter$ robots are located on a same node. When these $\mathcal{R}-1$ $righter$ robots execute Rule $\mathbf{M_6}$, one becomes a $potentialMin$, and the others become $dumbSearcher$. Therefore, by Observations 1 and 2 this rule can be executed only once. Moreover, by the rules of $\mathcal{GDG}$, once a $potentialMin$ stops to be a $potentialMin$, it cannot be again a $potentialMin$. Hence the lemma is proved. □

The following lemma demonstrates a property on min.

Lemma 9. *Before being min, r_{min} is either a righter robot or a potentialMin robot.*

Proof. A robot that is a min is a robot such that its variable $state$ is either equal to $minWaitingWalker$ or to $minTailWalker$. The only way to be a $minTailWalker$ robot is to be a $minWaitingWalker$ robot and to execute Rule $\mathbf{K_1}$. The only way to be a $minWaitingWalker$ is to execute Rule $\mathbf{M_1}$. Only $righter$ robots or $potentialMin$ robots can execute Rule $\mathbf{M_1}$ (refer to predicate $PotentialMinOrRighter()$). □

The three following lemmas give properties on the execution, when r_{min} is min. Indeed, they indicate the presence or absence of $righter/potentialMin$ in the execution while r_{min} is min.

Lemma 10. *In the suffix of the execution starting from the time where r_{min} is min, it is not possible to have a potentialMin robot and a righter robot present at the same time.*

Proof. By Lemma 9, r_{min} is either a $righter$ or a $potentialMin$ before being min. In the case where r_{min} is a $potentialMin$ before being min, then by Lemma 8, it cannot exist a $potentialMin$ in the execution after r_{min} becomes min. Therefore the lemma is proved in this case.

Consider now the case where r_{min} is a $righter$ before being min. For a robot to become a $potentialMin$ Rule $\mathbf{M_6}$ must be executed. This rule is executed when $\mathcal{R}-1$ $righter$ are located on a same node. While executing Rule $\mathbf{M_6}$, among the $\mathcal{R}-1$ $righter$ located on a same node, the one with the minimum identifier becomes $potentialMin$ while the others become $dumbSearcher$. By Observation 2, r_{min} cannot be among the $\mathcal{R}-1$ $righter$ that execute Rule $\mathbf{M_6}$, otherwise it cannot be a $righter$ before being min. Similarly thanks to Observation 2, the $\mathcal{R}-1$ robots that execute Rule $\mathbf{M_6}$, cannot be $righter$ anymore after the execution of this rule. therefore, it is not possible to have a $potentialMin$ and a $righter$ in the execution once r_{min} is min. □

Lemma 11. *If there exists a time t at which a righter, a robot r ($r \neq r_{min}$) such that $state_r \neq righter$ and r_{min}, as min, are present in the execution, then there is no more potentialMin in the suffix of the execution starting from t.*

Proof. By Lemma 10, since there is a $righter$ at time t, there is no $potentialMin$ in the execution at time t.

Since at time t, r_{min} and r are not $righter$ and can never be $righter$ anymore (refer to Observation 2), it is not possible to have $\mathcal{R}-1$ $righter$ located on a same node after time t. However, in order to have a $potentialMin$ in the execution, Rule $\mathbf{M_6}$ must be executed. This rule is executed only if $\mathcal{R}-1$ $righter$ are located on a same node. Therefore there is no $potentialMin$ in the execution after time t. □

Lemma 12. *If there is a potentialMin at a time t, and if before being min, r_{min} is a righter, then there is no more righter in the suffix of the execution starting from time $t' = max\{t, t_{min}\}$.*

Proof. Assume that before being min, r_{min} is a $righter$. Moreover assume that there is a $potentialMin$ in the execution at time t.

($*$) For a robot to become a $potentialMin$ Rule $\mathbf{M_6}$ must be executed. This rule is executed when $\mathcal{R}-1$ $righter$ are located on a same node. While executing Rule $\mathbf{M_6}$, among the $\mathcal{R}-1$ $righter$ located on a same node, the one with the minimum identifier becomes $potentialMin$ while the others become $dumbSearcher$. By Observation 2 none of these $\mathcal{R}-1$ robots can become $righter$ anymore after time t.

Consider then the two following cases.

Case 1: $\mathbf{t > t_{min}}$.

By Observation 2, r_{min} cannot be a *righter* after time t_{min}. Therefore r_{min} is not among the $\mathcal{R}-1$ robots that execute Rule $\mathbf{M_6}$, and hence, by $(*)$, after time t, there is no more *righter* in the execution.

Case 2: $\mathbf{t \leq t_{min}}$.

By $(*)$, r_{min} cannot be among the $\mathcal{R}-1$ *righter* that execute Rule $\mathbf{M_6}$, otherwise it cannot be a *righter* before being *min*. Therefore, by $(*)$ and since after time t_{min}, by Observation 2, r_{min} cannot be a *righter* anymore, there is no more *righter* in the execution after time t_{min}.

□

The following lemma is an extension of Lemma 4. While Lemma 4 is true only when all the robots are executing Phase `M`, the following lemma is true whether the robots are executing Phase `M` or Phase `K`.

Lemma 13. *If there is no $towerMin$ in the execution, if, at time t, a robot r is such that $state_r \in \{potentialMin, dumbSearcher, awareSearcher\}$, then, during the Move phase of time $t-1$, it does not consider the $\perp$ direction.*

Proof. Consider a robot r such that at time t, $state_r = potentialMin$. By Lemma 2, r considers the right direction during the Move phase of time $t-1$. Hence the lemma is proved in this case.

Consider now a robot r such that, at time t, $state_r \in \{dumbSearcher, awareSearcher\}$. Since there is no $towerMin$ in the execution, by the rules of $\mathcal{GDG}$ and knowing that initially all the robots are *righter*, there are only *righter*, $potentialMin$, *dumbSearcher*, *awareSearcher*, $waitingWalker$ and $minWaitingWalker$ robots in the execution. Note that there is no rule in $\mathcal{GDG}$ permitting a $waitingWalker$ or a $minWaitingWalker$ to become either a *dumbSearcher* or an *awareSearcher*.

Consider then the two following cases.

Case 1: At time $\mathbf{t-1}$, r is neither a *dumbSearcher* nor an *awareSearcher*.

Whatever the state of r at time $t-1$ (*righter* or $potentialMin$), to have its variable state at time t equals either to *dumbSearcher* or to *awareSearcher*, r executes at time $t-1$ either Rule $\mathbf{K_4}$, $\mathbf{M_5}$, $\mathbf{M_6}$ or $\mathbf{M_7}$.

When a robot executes Rule $\mathbf{K_4}$, it calls the function BecomeAwareSearcher. When a robot executes the function BecomeAwareSearcher, it sets its direction to the *right* direction, therefore the lemma is also true in this case.

Then, we can use the arguments of the proof of Lemma 4 to prove that the current lemma is true for the remaining cases. Indeed, even if in Lemma 4 the context is such that there is no *min* in the execution, the arguments used in its proof are still true in the context of the current lemma.

Case 2: At time $\mathbf{t-1}$, r is a *dumbSearcher* or an *awareSearcher*.

Whatever the state of r at time $t-1$ (*dumbSearcher* or *awareSearcher*), to have its variable state at time t equals either to *dumbSearcher* or to *awareSearcher*, r executes at time $t-1$ either Rule $\mathbf{M_9}$, $\mathbf{M_{10}}$ or $\mathbf{M_{11}}$. Similarly as for the case 1, we can use the arguments of the proof of Lemma 4 to prove that the current lemma is true in these cases.

□

The following lemma proves that in the case where there are at least 3 robots in the execution such that they are either $potentialMin$, *dumbSearcher* or *awareSearcher*, then, in finite time, at least one of this kind of robots is located on node u. A $potentialMin$, a *dumbSearcher* or an *awareSearcher* located with the $minWaitingWalker$ becomes a $waitingWalker$ (Rule $\mathbf{K_3}$). Therefore, this lemma permits to prove that in the case where there are at least 3 robots in the execution (after time t_{min}) such that they are either $potentialMin$, *dumbSearcher* or *awareSearcher*, then, in finite time, a supplementary $waitingWalker$ is located on node u.

To prove the following lemma, we need to introduce a new notion. We call $Seg(u, v)$ the set of nodes (of the footprint of the dynamic ring) between node u not included and v not included considering the right direction.

Lemma 14. *If there is no $towerMin$ in the execution but there exists at a time t at least 3 robots such that they are either $potentialMin$, dumbSearcher or awareSearcher, then it exists a time $t' \geq t$ at which at least a $potentialMin$, a dumbSearcher or an awareSearcher, reaches the node u.*

Proof. Assume that there is no $towerMin$ in the execution. By the rules of $\mathcal{GDG}$ and knowing that initially all the robots are *righter*, this implies that there are only *righter*, $potentialMin$, *dumbSearcher*, *awareSearcher*, $waitingWalker$ and $minWaitingWalker$ robots in the execution. Since there is no $towerMin$, r_{min} is $minWaitingWalker$ and is located on node u. By Lemma 7, we know that all the $waitingWalker$ robots (if

any) are on node u and r_{min} as well as the $waitingWalker$ robots do not move. This implies that among the robots that are not on node u there are only $righter$, $potentialMin$, $dumbSearcher$ and $awareSearcher$.

Assume by contradiction that at a time t, there are at least 3 robots such that they are either $potentialMin$, $dumbSearcher$ or $awareSearcher$ and such that for all time $t' \geq t$ none of these kinds of robots succeed to reach the node u at time t'. We consider the execution from time t.

Consider a robot r such that at time t, $state_r \in \{potentialMin, dumbSearcher, awareSearcher\}$.

(i) If r is an $awareSearcher$, since it cannot reach u, it executes Rule $\mathbf{M_{11}}$, and hence it executes the function SEARCH. The variable state of r is not updated while r executes this function, therefore r is an $awareSearcher$ and executes Rule $\mathbf{M_{11}}$ and the function SEARCH at each instant time from time t. Thus by Lemma 13, r always considers a direction different from $\perp$ after time t included.

(ii) If r is a $dumbSearcher$, since it cannot reach u, it can execute either Rule $\mathbf{M_{10}}$ (if it is on the same node as an $awareSearcher$) and hence becomes an $awareSearcher$ robot and executes the function SEARCH, Rule $\mathbf{M_9}$ (if it is on the same node as a $righter$) and hence becomes an $awareSearcher$ and executes the function SEARCH, or Rule $\mathbf{M_{11}}$ and hence stays a $dumbSearcher$ and executes the function SEARCH. By Lemma 13 and by (i), r always considers a direction different from $\perp$ after time t included.

(iii) If r is a $potentialMin$, by Lemma 10, there is no $righter$ in the execution at time t and therefore by Observation 2 there is no $righter$ in the execution after time t included. Therefore, since r cannot reach u, it can execute either Rule $\mathbf{M_5}$ (if it is on the same node as an $awareSearcher$) and hence becomes an $awareSearcher$ and executes the function SEARCH, or Rule $\mathbf{M_8}$ and hence stays a $potentialMin$ and considers the right direction. Therefore by Lemma 13 and by (i), r always considers a direction different from $\perp$ after time t included.

(iv) If there is a $righter$ robot in the execution after time t, then by Lemma 11, there is no $potentialMin$ robot in the execution after time t included. If a $righter$ robot is on the same node as a $dumbSearcher$ or as an $awareSearcher$, it executes Rule $\mathbf{M_7}$ and hence becomes an $awareSearcher$ and executes the function SEARCH.

(v) While executing the function SEARCH if a robot is isolated it considers the last direction it considered. While executing the function SEARCH if a robot is not isolated, if it possesses the maximum identifier among all the robots of its current location it considers the left direction otherwise it considers the right direction.

(vi) Note that if there is a $potentialMin$ while r_{min} is $minWaitingWalker$, then it possesses the minimum identifier among all the robots not located on node u. Indeed, only Rule $\mathbf{M_6}$ permits a robot to become a $potentialMin$. For this rule to be executed, $\mathcal{R}-1$ $righter$ robots must be located on a same node. While executing this rule the robot with the minimum identifier among the $\mathcal{R}-1$ robots located on a same node becomes $potentialMin$. Since, by Lemma 8, there is only one $potentialMin$ in the whole execution and since by definition r_{min} possesses the minimum identifier among all the robots of the system, r_{min} does not execute Rule $\mathbf{M_6}$. Therefore, while r_{min} is $minWaitingWalker$, the $potentialMin$ possesses the minimum identifier among all the robots not located on node u. Thus, when a $potentialMin$ executes Rule $\mathbf{M_8}$ and hence considers the right direction it possesses the same behavior as if it was executing the function SEARCH.

Case 1: There is no eventual missing edge.

Call d the direction of r during the Look phase of time t, and let v be the node where r is located during the Look phase of time t. Call w the adjacent node of v in the direction d. Call e the edge between v and w. As proved in cases (i), (ii) and (iii), d is either equal to $right$ or $left$.

We want to prove that it exists a time t' ($t' \geq t$) such that a robot r' (it is possible to have $r' = r$) with $state_{r'} \in \{potentialMin, dumbSearcher, awareSearcher\}$ considers the direction d and is located on w during the Look phase of time t'.

Call t" (t" $\geq t$) the first time after time t included where there is an adjacent edge to v. If during the Move phase of time t", r does not consider the direction d, by $(i)-(vi)$ this necessarily implies that when r executes the function SEARCH (or a function that behaves like the function SEARCH) there is at least another robot on its node. Moreover by $(i)-(vi)$ the other robot(s) with r also executes the function SEARCH (or a function that behaves like the function SEARCH) and is or becomes $potentialMin$, $dumbSearcher$ or $awareSearcher$. Therefore, since all the robots possess distinct identifiers and by (v), during the Move phase of time t", a robot among $\{potentialMin, dumbSearcher, awareSearcher\}$ on node v considers the direction d.

Since all the edges are infinitely often present we can repeat these arguments on each instant time until the time t_e where e is present. At time t_e a robot (either $potentialMin$, $dumbSearcher$, $awareSearcher$) considers the direction d and hence crosses e. Since the direction considered by a robot can be updated only during Compute phases, we succeed to prove that t' exists.

Applying these arguments recurrently we succeed to prove that in finite time a robot r" such that $state_{r"} \in \{potentialMin, dumbSearcher, awareSearcher\}$ is on node u.

Case 2: There is an eventual missing edge.

Call e the eventual missing edge. Consider the execution after the time greater or equal to t where e is missing forever. Call v the node such that its adjacent right edge is e. Call w the adjacent right node of v.

At least two robots that are either *potentialMin*, *dumbSearcher* or *awareSearcher* are either on nodes in $Seg(u, v) \cup \{v\}$ or on nodes in $Seg(w, u) \cup \{w\}$.

Assume that there are at least two robots that are either *potentialMin*, *dumbSearcher* or *awareSearcher* which are on nodes in $Seg(u, v) \cup \{v\}$. The reasoning when there are at least two robots that are either *potentialMin*, *dumbSearcher* or *awareSearcher* which are on nodes in $Seg(w, u) \cup \{w\}$ is similar.

The edge e is an eventual missing edge. It can exist only one eventual missing edge in $\mathcal{COT}$ ring. Therefore all the edges between the nodes in $\{u\} \cup Seg(u, v) \cup \{v\}$ are infinitely often present. Thus, if there exists a robot (either *potentialMin*, *dumbSearcher* or *awareSearcher*) that considers the left direction then we can apply the arguments of case 1 to prove that in finite time a robot r", such that $state_{r"} \in \{potentialMin, dumbSearcher, awareSearcher\}$, is on node u.

Therefore consider that all the robots, that are either *potentialMin*, *dumbSearcher* or *awareSearcher* and that are located on nodes in $Seg(u, v) \cup \{v\}$, consider the right direction. In this case a robot either *potentialMin*, *dumbSearcher* or *awareSearcher* cannot be located on the same node as a robot either *righter*, *potentialMin*, *dumbSearcher* or *awareSearcher*, otherwise during the Move phase of the time of this meeting, by $(i)-(vi)$, it exists a robot either *potentialMin*, *dumbSearcher* or *awareSearcher* that considers the left direction.

Since e is an eventual missing edge, and since there are at least two robots either *potentialMin*, *dumbSearcher* or *awareSearcher* that consider the right direction, applying the arguments of case 1 on two of these robots, we succeed to prove that in finite time two of these robots are located on v. Therefore, by the previous paragraph, in finite time a robot r", such that $state_{r"} \in \{potentialMin, dumbSearcher, awareSearcher\}$, is on node u.

□

Now, we prove one of the two main lemmas of this phase: we prove that if a *potentialMin* is present in the execution, then, in finite time, a *towerMin* is present in the execution. While proving this lemma, we also prove that, at the time when the *towerMin* is formed, among the two robots not involved in this *towerMin*, it can exit at most one *righter*. This information is useful to prove Phase T.

Lemma 15. *If there is a potentialMin in the execution, then there exists a time t at which a towerMin is present and among the robots not involved in the towerMin there is at most one righter robot at time t.*

Proof. Assume that there exists a time t at which a *potentialMin* robot is present in the execution. Assume by contradiction that there is no *towerMin* in the execution. In the following, we consider the execution from time $t' = max\{t, t_{min}\}$.

Since there is no *towerMin*, by the rules of $\mathcal{GDG}$ and knowing that initially all the robots are *righter*, there are in the execution only *righter*, *potentialMin*, *dumbSearcher*, *awareSearcher*, *waitingWalker* and *minWaitingWalker* robots. By Lemma 7, all the *waitingWalker* are located on the same node as r_{min}, when $state_{r_{min}} = minWaitingWalker$, and both r_{min}, as a *minWaitingWalker*, and the *waitingWalker* robots do not move. By Corollary 4, only r_{min} can be a *minWaitingWalker*. We recall that r_{min} as *minWaitingWalker* is located on node u. Therefore the *minWaitingWalker* and all the *waitingWalker* (if any) are located on node u.

By Lemma 9 we know that, before being *min*, r_{min} is either a *righter* robot or a *potentialMin* robot. We can then consider the two following cases.

Case 1: Before being min, $\mathbf{r_{min}}$ is a righter robot.

By Lemma 12, at time t' there are only *potentialMin*, *dumbSearcher*, *awareSearcher*, *waitingWalker* and *minWaitingWalker* robots in the execution. Moreover, in this case, all the robots that are not on node u are necessarily either *potentialMin*, *dumbSearcher* or *awareSearcher*.

When a *potentialMin*, a *dumbSearcher* or an *awareSearcher* robot meets the *minWaitingWalker*, it executes Rule $\mathbf{K_3}$, hence it becomes a *waitingWalker* and stops to move.

Then each time there are at least 3 robots in the execution such that they are either *potentialMin*, *dumbSearcher* and/or *awareSearcher*, using Lemma 14, we succeed to prove that at least one *potentialMin*, *dumbSearcher* or *awareSearcher* succeeds to join the node u and therefore becomes a *waitingWalker*. Therefore, by Lemma 7, a *towerMin* is formed in finite time.

Case 2: Before being min, $\mathbf{r_{min}}$ is a potentialMin.

For a robot to become a *potentialMin*, Rule $\mathbf{M_6}$ must be executed. This rule is executed when $\mathcal{R}-1$ *righter* are located on a same node. While executing this rule, among the $\mathcal{R}-1$ *righter* located on a same node, one becomes *potentialMin* while the other become *dumbSearcher*. By Observation 2, none of these $\mathcal{R}-1$ robots can become *righter* anymore. Therefore, by Lemma 8, once r_{min} is *min*, there are only *dumbSearcher*, *awareSearcher*, *waitingWalker*, *minWaitingWalker* robots and at most one *righter* robot in the execution. Moreover, in this case, among the robots that are not on node u, there are only *dumbSearcher* and *awareSearcher* and at most one *righter*. By the rules of $\mathcal{GDG}$, as long as a *dumbSearcher* or an *awareSearcher* is not on node u, its variable state stays in $\{dumbSearcher, awareSearcher\}$.

Once r_{min} is *min*, if there exists a time at which there is no more *righter* robot in the execution, then, using the arguments of case 1, we succeed to prove that a *towerMin* is formed in finite time. Therefore assume that there is always a *righter* robot r in the execution.

When a *dumbSearcher* or an *awareSearcher* robot is located on the same node as the *minWaitingWalker*, it executes Rule $\mathbf{K_3}$, hence it becomes a *waitingWalker* and stops to move. Then, using multiple times Lemma 14 and Lemma 7, we know that in finite time there are in the execution only one *righter* and only 2 robots r' and r" such that $state_{r'}, state_{r"} \in \{awareSearcher, dumbSearcher\}^2$ (all the other robots are *minWaitingWalker* and *waitingWalker* robots and are located on node u). Note that r' (resp. r") cannot be located on node u, otherwise, by Rule $\mathbf{K_3}$, a *towerMin* is formed. Therefore, r' and r" always have their variable state in $\{dumbSearcher, awareSearcher\}$.

When a *righter* robot is located on the same node as an *awareSearcher* or as a *dumbSearcher*, it executes Rule $\mathbf{M_7}$ and becomes an *awareSearcher*. Similarly, if a *righter* is on the same node as a *minWaitingWalker* while the adjacent right edge to its position is present, then the *righter* robot executes Rule $\mathbf{K_4}$ and becomes an *awareSearcher*. Therefore, as highlighted previously, these situations cannot happen, otherwise a *towerMin* is formed in finite time. This implies that, as long as the robot r is not on node u, it must be isolated. Since r' and r" cannot be located on node u, if r succeeds to join the node u in the case there is no present adjacent right edge to u, then r executes Rule $\mathbf{M_8}$ and therefore stays a *righter* and considers the right direction. Therefore, since an isolated *righter* robot always executes Rule $\mathbf{M_8}$, hence always considers the right direction, this implies that either r is on a node v ($v \neq u$) such that the adjacent right edge of v is an eventual missing edge at least from the time where r is on node v (case 2.1) or r succeeds to reach u but the adjacent right edge of u is an eventual missing edge at least from the time where r is on node u (case 2.2).

($*$) When an *awareSearcher* or a *dumbSearcher* is isolated it executes Rule $\mathbf{M_{11}}$, hence executes the function SEARCH, therefore it considers the last direction it considered. By Lemma 13, this direction cannot be equal to $\perp$.

($**$) Since only r' and r" have their variable state in $\{dumbSearcher, awareSearcher\}^2$, and since r' and r" cannot be located on node u and cannot be located with r, if a *dumbSearcher* is located on the same node as an *awareSearcher* or if an *awareSearcher* (resp. a *dumbSearcher*) is located on the same node as another *awareSearcher* (resp. *dumbSearcher*), necessarily this means that r' and r" are located on a same node, and there is no other robot on the same node as them. When a *dumbSearcher* is on the same node as an *awareSearcher* it executes Rule $\mathbf{M_{10}}$, hence it becomes an *awareSearcher* and executes the function SEARCH. When an *awareSearcher* is on the same node as a *dumbSearcher* it executes Rule $\mathbf{M_{11}}$ and hence executes the function SEARCH. Since r' and r" have distinct identifiers, when an *awareSearcher* and a *dumbSearcher* are on a same node, they both execute the function SEARCH, therefore one considers the right direction, while the other one considers the left direction. Similarly, if two *awareSearcher* (resp. *dumbSearcher*) robots are on the same node, they both execute Rule $\mathbf{M_{11}}$ and hence the function SEARCH, therefore one considers the right direction, while the other one considers the left direction.

Case 2.1: Let w be the adjacent node of v in the right direction. It can exist only one eventual missing edge, which is the adjacent right edge of node v. Therefore, if a robot, in $Seg(u, v)$ or in $Seg(w, u)$, considers a direction d and does not change this direction, it eventually succeeds to move in this direction. Similarly, if a robot is on node w and always considers the right direction, it eventually succeeds to move in this direction ($***$).

Firstly, assume that only r' (resp. r") is on a node in $Seg(u, v)$. By ($*$) and ($***$), r' (resp. r") cannot consider the right direction, otherwise it reaches r in finite time. Therefore r' (resp. r") considers the left direction. By ($*$) and ($***$), in finite time, r' (resp. r") succeeds to reach u, implying that a *towerMin* is formed.

Secondly, assume that r' and r" are on nodes in $Seg(u, v)$. By $(*)$, $(**)$ and $(***)$, they cannot meet otherwise one of them reaches u in finite time. Moreover, if they do not meet none of them can consider the left direction otherwise, by $(*)$ and $(***)$, they reach u in finite time. Therefore, they cannot meet and must consider the right direction. By $(*)$ and $(***)$, in finite time one robot among r' and r" succeeds to reach r, implying that a $towerMin$ is formed.

Thirdly, assume that r and r" are on nodes in $Seg(v, u)$. By $(*)$, $(**)$ and $(***)$, they cannot meet otherwise one of them reaches u in finite time. Moreover, if they do not meet none of them can consider the right direction otherwise, by $(*)$ and $(***)$, they reach u in finite time. Therefore, they cannot meet and must consider the left direction. However, by $(*)$ and $(***)$, since the adjacent right edge of v is missing forever, in finite time r' and r" reach w, which is a contradiction with the fact that they do not meet.

Case 2.2: Applying the arguments used in the case 2.1, when r' and r" are on nodes in $Seg(v, u)$, to r' and r" when there are on nodes in $Seg(u, u)$, we succeed to prove that in finite time at least one of them reaches node u, making Rule $\mathbf{Term_2}$ true, which leads to a contradiction.

□

Finally, we prove the other main lemma of this phase: we prove that even if there is no $potentialMin$ in the execution, then, in finite time, a $towerMin$ is present in the execution. While proving this lemma, we also prove that, at the time when the $towerMin$ is formed, among the two robots not involved in this $towerMin$, it can exit at most one $righter$. This information is useful to prove Phase T.

Lemma 16. *If there is no potentialMin in the execution, then there exists a time t at which a towerMin is present and among the robots not involved in the towerMin there is at most one righter robot at time t.*

Proof. Assume, by contradiction, that there is no $towerMin$ in the execution. By the rules of $\mathcal{GDG}$ and knowing that initially all the robots are $righter$, this implies that there are only $righter$, $potentialMin$, $dumbSearcher$, $awareSearcher$, $waitingWalker$ and $minWaitingWalker$ robots in the execution.

Assume that there is no $potentialMin$ in the execution. If there is no $potentialMin$ in the execution, it cannot exist $dumbSearcher$ in the execution. Indeed, the only way for a robot to become $dumbSearcher$ is to execute Rule $\mathbf{M_6}$. However, when this rule is executed, a robot becomes $potentialMin$. Therefore, there are in the execution only $righter$, $awareSearcher$, $waitingWalker$ and $minWaitingWalker$ robots.

Before time t_{min}, by the rules of $\mathcal{GDG}$, there are only $righter$ in the execution. Indeed, by Corollary 4, only r_{min} can be $minWaitingWalker$ and it becomes $minWaitingWalker$ at time t_{min}. Moreover, the only way for a robot to become $waitingWalker$ is to execute Rule $\mathbf{K_3}$. In the case where there is no $potentialMin$ in the execution, only an $awareSearcher$ located with r_{min}, as a $minWaitingWalker$, can execute this rule. Besides, the only ways for a robot to become an $awareSearcher$ is either to be a $righter$ and to be located with an $awareSearcher$ (refer to Rule $\mathbf{M_7}$), or to be a $righter$ and to be located with r_{min}, as a $minWaitingWalker$, while an adjacent right edge to their location is present (refer to Rule $\mathbf{K_4}$). Since initially all the robots are $righter$, the first $awareSearcher$ of the execution can be present only thanks to the execution of Rule $\mathbf{K_4}$.

All this implies that, even after time t_{min}, as long as no $righter$ robot is on node u with r_{min}, as a $minWaitingWalker$, while there is a present adjacent right edge to u, it cannot exist neither $awareSearcher$ nor $waitingWalker$ in the execution: there is at most one $minWaitingWalker$ and there are at least $\mathcal{R}-1$ $righter$. Moreover, this implies that as long as the situation described has not happened, all the $righter$ robots only execute Rule $\mathbf{M_8}$, hence always consider the right direction.

Consider the execution just after time t_{min}. In this context, necessarily, in finite time, there exists a $righter$ robot r that succeeds to reach u (while r_{min} is $minWaitingWalker$). Indeed, if this is not the case, this implies that there exists an eventual missing edge e. Since all the $righter$ robots always consider the right direction and since it can exist at most one eventual missing edge, this implies that $\mathcal{R}-1$ $righter$ robots reach in finite time the same extremity of e. Thus, Rule $\mathbf{M_6}$ is executed, which leads to a contradiction with the fact that there is no $potentialMin$ in the execution.

Similarly, necessarily, in finite time, there exists an adjacent right edge to u while r is on u. Indeed, if this is not the case, this implies that the adjacent right edge of u is an eventual missing edge. Since all the $righter$ robots always consider the right direction and since it can exist at most one eventual missing edge, in finite time all the $righter$ succeed to be located on node u. This implies that Rule $\mathbf{Term_1}$ is executed, which leads to a contradiction.

Therefore there exists a time t' at which r executes Rule $\mathbf{K_4}$. At this time r becomes an $awareSearcher$ robot and considers the right direction. We then consider the execution from time t'.

$(*)$ From this time t', as long as there exists $righter$ in the execution, it always exists an $awareSearcher$ robot r' considering the right direction, such that there is no $righter$ robots on $Seg(u, v)$, where v is the node where r' is currently located. This can be proved by analyzing the movements of the different kinds of robots that we describe in $(i) - (vii)$.

(i) By Lemma 7, all the $minWaitingWalker$ and $waitingWalker$ (if any) are on a same node (which is the node u) and do not move.

(ii) If an $awareSearcher$ is located on node u, therefore if it is located with r_{min}, as a $minWaitingWalker$, it executes Rule $\mathbf{K_3}$ and becomes a $waitingWalker$ robot.

(iii) If an $awareSearcher$ is on a node different from the node u, the only rule it can execute is Rule $\mathbf{M_{11}}$, in which the function SEARCH is called. While executing this function, an isolated $awareSearcher$ considers the direction it considers during its last Move phase. By Lemma 13, this direction cannot be $\perp$.

(iv) If a $righter$ robot is located only with other $righter$ robots or if it is located on node u, therefore if it is located with r_{min}, as a $minWaitingWalker$, such that there is no adjacent right edge to u, it executes Rule $\mathbf{M_8}$, hence it stays a $righter$ and considers the right direction.

(v) If a $righter$ robot is with r_{min}, as a $minWaitingWalker$, such that there is an adjacent right edge to u, then it executes Rule $\mathbf{K_4}$ and hence becomes an $awareSearcher$.

(vi) If a $righter$ robot is on a node different from node u with an $awareSearcher$, it executes Rule $\mathbf{M_7}$ and therefore becomes $awareSearcher$ and executes the function SEARCH.

(vii) Note that by the movements described in (i) to (vi), if a robot executes the function SEARCH, then all the robots that are on the same node as it also execute this function. While executing the function SEARCH, if multiple robots are on the same node, one considers the left direction, while the others consider the right direction.

Applying these movements on r' and recursively on the robots that r' meet that consider the right direction after their meeting with r' and so on, we succeed to prove the property ($*$).

($**$) Note that if there exists a time at which there is no more $righter$ in the execution, then by applying (ii), Lemma 7 and Lemma 14 multiple times we succeed to prove that a $towerMin$ is formed. Therefore at least one robot is always a $righter$ during the whole execution. Call $\mathcal{S}_r$ the set of $righter$ robots that stay $righter$ during the whole execution.

Let us consider the following cases.

Case 1: There does not exist an eventual missing edge.

None of the robots of $\mathcal{S}_r$ can be located on the same node as an $awareSearcher$, otherwise, by (vi), they become $awareSearcher$. Therefore, all the robots of $\mathcal{S}_r$ that are not on node u can only consider the right direction (refer to (iv)). Since all the edges are infinitely often present, for each robot r" of $\mathcal{S}_r$, it exists a time at which r" is on node u. Moreover, once on node u, as long as there is no adjacent right edge to u, r" considers the right direction (refer to (iv)), and therefore stays on node u. Thus, since all the edges are infinitely often present, for each robot r" of $\mathcal{S}_r$, it exists a time at which r" is on node u such that an adjacent right edge to u is present. Therefore, by (v), in finite time, all the robots of $\mathcal{S}_r$ are $awareSearcher$ robots. Hence, by ($**$), the lemma is proved.

Case 2: There exists an eventual missing edge.

Call x the node such that its adjacent right edge is the eventual missing edge. Consider the execution after time t' such that the eventual missing edge is missing forever.

Case 2.1: x = u.

None of the robots of $\mathcal{S}_r$ can be located on the same node as an $awareSearcher$, otherwise, by (vi), they become $awareSearcher$. Therefore, all the robots of $\mathcal{S}_r$ that are not on node u can only consider the right direction (refer to (iv)). Since it can exist at most one eventual missing edge, in finite time the robots of $\mathcal{S}_r$ succeed to reach node u, and stay on node u (refer to (iv)). Necessarily, $|\mathcal{S}_r| < \mathcal{R}-2$, otherwise Rule $\mathbf{Term_2}$ is executed. At the time at which all the robots of $\mathcal{S}_r$ are on node u, by ($*$), we know that at least one $awareSearcher$, on a node v, considers the right direction. By (vi), none of the $righter$ of $\mathcal{S}_r$ can be located on node v. Therefore, this $awareSearcher$ is not on node u. By the movements described in (iii) and (vii), we know that in finite time an $awareSearcher$ succeeds to reach node u. Then all the $righter$ of $\mathcal{S}_r$ become $awareSearcher$, hence by ($**$), the lemma is proved.

Case 2.2: x $\neq$ u.

None of the robots of $\mathcal{S}_r$ can be located on the same node as an $awareSearcher$, otherwise, by (vi), they become $awareSearcher$. Therefore, none of the robots of $\mathcal{S}_r$ can be located on $Seg(u,x) \cup \{x\}$, otherwise, in finite time, by (iv) they are located on node x. However, once all the robots of $\mathcal{S}_r$ are on node x, by ($*$), and the movements described in (iii) and (vii) an $awareSearcher$ succeeds to be located on node u in finite time, which leads to a contradiction. Therefore all the robots of $\mathcal{S}_r$ are on nodes in $Seg(x,u)$. Since it can exist only one eventual missing edge, and since this edge is the adjacent right edge of x, for each robot r" of $\mathcal{S}_r$, by (iv), it exists a time at which r" is on node u while there is a present adjacent right edge to u. Therefore, by (v), in finite time all the robots of $\mathcal{S}_r$ are $awareSearcher$ robots. Hence, by ($**$), the lemma is proved.

We just proved that it exists a time t_{tower} at which a $towerMin$ is present in the execution. We now prove that, at time t_{tower}, among the robots not involved in the $towerMin$, there is at most one $righter$. By Lemma 6, there is only one $towerMin$ in the whole execution. Necessarily, as explained above when there is no $potentialMin$ in the execution, in order to have a $towerMin$, a $righter$ must become an $awareSearcher$ while executing Rule $\mathbf{K_4}$. The property $(*)$ is then true. By definition of a $towerMin$, only two robots are not involved in the $towerMin$. Assume, by contradiction, that there are two $righter$ not involved in the $towerMin$ at time t_{tower}. By $(*)$, this implies that there is an $awareSearcher$ at time t_{tower}. However, by definition, a $towerMin$ is composed of one $minWaitingWalker$ and $\mathcal{R}-3$ $waitingWalker$, therefore, since there are $\mathcal{R}$ robots in the system and among them, at time t_{tower}, two are $righter$ and one is an $awareSearcher$, there is a contradiction with the fact that there is a $towerMin$ at time t_{tower}. □

By Lemmas 15 and 16, we can deduce the following corollary which proves the correctness of Phase K.

Corollary 5. *There exists a time t in the execution at which a $towerMin$ is present and among the robots not involved in the $towerMin$ there is at most one $righter$ robot at time t.*

5.1.3 Proofs of Correctness of Phases W and T

The combination of Phases W and T of $\mathcal{GDG}$ permit to solve $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings. Since $\mathbb{G}_{EW}$ is divided into a safety and a liveness property, to prove the correctness of Phases W and T, we have to prove each of these two properties. We recall that, to satisfy the safety property of the gathering problem, all the robots that terminate their execution have to do so on the same node, and to satisfy the liveness property of $\mathbb{G}_{EW}$, at least $\mathcal{R}-1$ robots must terminate their execution in finite time. In this subsection, we, first, prove that $\mathcal{GDG}$ solves the safety of the gathering problem in $\mathcal{COT}$ rings, and then, we prove that $\mathcal{GDG}$ solves the liveness of $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings. We prove this respectively in Lemmas 19 and 21. To prove these two lemmas, we need to prove some other lemmas.

By Corollary 5, we know that, in finite time, a $towerMin$ is formed. By Lemma 6, there is at most one $towerMin$ in the execution. Therefore, there is one and only one $towerMin$ in the execution. Call T such a $towerMin$. Let t_{tower} be the time at which T is formed. By definition, a $towerMin$ is composed of $\mathcal{R}-2$ robots. Call r_1 and r_2 the two robots that are not involved in T.

In the previous subsection, we prove that, at time t_{tower}, at most one of the robots among r_1 and r_2 is a $righter$. In the following lemma, we go farther and give the set of possible values for the variable $state$ at time t_{tower} of each of these robots.

Lemma 17. *At time t_{tower}, $state_{r_1} \in \{righter, potentialMin, dumbSearcher, awareSearcher\}$ and $state_{r_2} \in \{dumbSearcher, awareSearcher\}$.*

Proof. Until the Look phase of time t_{tower}, by the rules of $\mathcal{GDG}$ and knowing that all the robots are initially $righter$, there are only $righter$, $potentialMin$, $dumbSearcher$, $awareSearcher$, $waitingWalker$ and $minWaitingWalker$ robots in the execution.

By Corollary 4, only r_{min} can be min, therefore only r_{min} can be $minWaitingWalker$. By definition of a $towerMin$, a $minWaitingWalker$ is involved in T. Since r_1 and r_2 are not involved in T, this implies that neither r_1 nor r_2 can be $minWaitingWalker$ at time t_{tower}.

By definition of a $towerMin$, at time t_{tower}, the $\mathcal{R}-2$ robots involved in T are on a same node. This node is the node u. Therefore, at time t_{tower} neither r_1 nor r_2 can be located on node u, otherwise Rule $\mathbf{Term_2}$ is executed. By Lemma 7, this implies that neither r_1 nor r_2 can be a $waitingWalker$ at time t_{tower}.

By Corollary 5, at time t_{tower}, only one robot among r_1 and r_2 can be a $righter$ robot. Assume without lost of generality that r_1 is a $righter$ at time t_{tower}. In this case by Corollary 5, r_2 cannot be a $righter$ at time t_{tower}. Moreover, in this case, by Lemma 10, r_2 cannot be a $potentialMin$ at time t_{tower}.

Now assume without lost of generality that r_1 is a $potentialMin$ robot at time t_{tower}. By Lemmas 8 and 10, r_2 can neither be a $potentialMin$ nor a $righter$ at time t_{tower}.

This prove the lemma. □

In the following lemma, we prove a property on Rules $\mathbf{Term_1}$ and $\mathbf{Term_2}$ that helps us to prove that $\mathcal{GDG}$ solves $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings.

Lemma 18. *If a robot r, on a node x, at a time t, executes Rule $\mathbf{Term_1}$ (resp. $\mathbf{Term_2}$), then there are $\mathcal{R}$ (resp. $\mathcal{R}-1$) robots on node x at time t and they all execute Rule $\mathbf{Term_1}$ (resp. $\mathbf{Term_2}$) at time t (if they are not already terminated).*

Proof. If a robot r, on a node x, executes Rule $\mathbf{Term_1}$ (resp. $\mathbf{Term_2}$) at a time t, by the predicate $\mathbb{G}_E()$ (resp. $\mathbb{G}_{EW}()$), there are $\mathcal{R}$ (resp. $\mathcal{R}-1$) robots on x at time t. Moreover, if the predicate $\mathbb{G}_E()$ (resp. $\mathbb{G}_{EW}()$) is true for r at time t, since the robots are fully-synchronous, it is necessarily true for all the robots (not already terminated) located on node x at time t. This implies that all the robots (not already terminated) located on x at time t, execute Rule $\mathbf{Term_1}$ (resp. $\mathbf{Term_2}$) at time t. □

Now we prove one of the two main lemmas of this subsection: we prove that $\mathcal{GDG}$ solves the safety property of the gathering problem in $\mathcal{COT}$ rings.

Lemma 19. *$\mathcal{GDG}$ solves the safety of the gathering problem in $\mathcal{COT}$ rings.*

Proof. We want to prove that, while executing $\mathcal{GDG}$, all robots that terminate their execution terminate it on the same node. While executing $\mathcal{GDG}$, the only way for a robot to terminate its execution is to execute either Rule $\mathbf{Term_1}$ or Rule $\mathbf{Term_2}$.

By Lemma 18, if a robot r, on a node x, at a time t, executes Rule $\mathbf{Term_1}$, then there are $\mathcal{R}$ robots on node x at time t and they all execute Rule $\mathbf{Term_1}$ at time t (if they are not already terminated). Therefore, in the case where r executes Rule $\mathbf{Term_1}$ at time t, all the robots of the system are terminated on x at time t, hence the lemma is proved in this case.

By Lemma 18, if a robot r, on a node x, at a time t, executes Rule $\mathbf{Term_2}$, then there are $\mathcal{R}-1$ robots on node x at time t and they all execute Rule $\mathbf{Term_2}$ at time t (if they are not already terminated). Therefore, in the case where r executes Rule $\mathbf{Term_2}$ at time t, $\mathcal{R}-1$ robots of the system are terminated on x at time t. Call r' the robot that is not on the node x at time t. Let y ($y \neq x$) be the node where r is located at time t. To prove the lemma, it stays to prove that r' is not terminated at time t, and that after time t, r' either terminates its execution on node x or never terminates its execution.

Assume, by contradiction, that at time t, r' is terminated. This implies that there exists a time $t' \leq t$ at which r' executes either Rule $\mathbf{Term_1}$ or Rule $\mathbf{Term_2}$. By Lemma 18, this implies that at least $\mathcal{R}-2$ other robots are terminated on node y at time t'. Therefore, there is a contradiction with the fact that r executes Rule $\mathbf{Term_2}$ at time t on node x. Indeed, to execute Rule $\mathbf{Term_2}$ at time t on node x, $\mathcal{R}-1$ robots must be located on node x at time t, since $\mathcal{R} \geq 4$, it is not possible to have $\mathcal{R}-1$ robots on node x at time t.

Moreover, after time t, by Lemma 18, r' can terminate its execution only on node x (since it is the only node where $\mathcal{R}-1$ robots are located). Therefore, the lemma is proved. □

The following lemma is an extension of Lemma 13. While Lemma 13 is true when the robots are either executing Phase `M` or Phase `K`, the following lemma is true whatever the phase of the algorithm the robots are executing.

Lemma 20. *If, at time t, an isolated robot r is such that $state_r \in \{dumbSearcher, awareSearcher\}$, then, during the Move phase of time $t-1$, it does not consider the $\perp$ direction.*

Proof. By the rules of $\mathcal{GDG}$, $minWaitingWalker$, $waitingWalker$, $minTailWalker$, $tailWalker$, $headWalker$ and $leftWalker$ cannot become $dumbSearcher$ or $awareSearcher$.

Consider an isolated robot r such that, at a time t, $state_r \in \{dumbSearcher, awareSearcher\}$.

Consider then the two following cases.

Case 1: At time t − 1, r is neither a *dumbSearcher* nor an *awareSearcher*.

Whatever the state of r at time $t-1$ ($righter$ or $potentialMin$), to have its variable state at time t equals either to $dumbSearcher$ or to $awareSearcher$, r executes at time $t-1$ either Rule $\mathbf{K_4}$, $\mathbf{M_2}$, $\mathbf{M_3}$, $\mathbf{M_5}$, $\mathbf{M_6}$ or $\mathbf{M_7}$.

When a robot executes Rule $\mathbf{M_2}$, it calls the function BecomeAwareSearcher. When a robot executes the function BecomeAwareSearcher, it sets its direction to the $right$ direction, therefore the lemma is true in this case.

A robot executes Rule $\mathbf{M_3}$ only if it is located with a $headWalker$ on a node x. Necessarily there is no present adjacent right edge to x at time $t-1$, otherwise the robot would have executed Rule $\mathbf{M_2}$. By the rules of $\mathcal{GDG}$, a $headWalker$ only considers the $\perp$ direction or the right direction. Indeed, a $headWalker$ can only execute Rules $\mathbf{T_2}$, $\mathbf{T_3}$ and $\mathbf{W_1}$. While executing Rule $\mathbf{T_2}$, a $headWalker$ becomes a $leftWalker$ and considers the $\perp$ direction. While executing Rule $\mathbf{T_3}$, a $headWalker$ considers the $\perp$ direction. Finally, while executing Rule $\mathbf{W_1}$, a $headWalker$ considers either the right direction or the $\perp$ direction. Therefore, even if, after the execution of Rule $\mathbf{M_3}$, r considers the $\perp$ direction, it is not isolated at time t, hence the lemma is not false in this case.

Then, we can use the arguments of the proof of Lemma 13 (in the case where the robot r is a $dumbSearcher$ or an $awareSearcher$ at time t) to prove that the current lemma is true for the remaining cases. Indeed, even if in Lemma 13 the context is such that there is no $towerMin$ in the execution, the arguments used in its proof are still true in the context of the current lemma.

Case 2: At time t − 1, r is a *dumbSearcher* or an *awareSearcher*.

Whatever the state of r at time $t-1$ ($dumbSearcher$ or $awareSearcher$), to have its variable state at time t equals either to $dumbSearcher$ or to $awareSearcher$, r executes at time $t-1$ either Rule $\mathbf{M_2}$, $\mathbf{M_3}$, $\mathbf{M_9}$, $\mathbf{M_{10}}$ or $\mathbf{M_{11}}$.

We can use the arguments of Case 1 to prove that while executing Rule $\mathbf{M_2}$ or $\mathbf{M_3}$, the lemma is proved.

Then, similarly as for the Case 1, we can use the arguments of the proof of Lemma 13 (in the case where the robot r is a *dumbSearcher* or an *awareSearcher* at time t) to prove that the current lemma is true in the remaining cases of Case 2.

□

Finally, we prove the other main lemma of this subsection: we prove that $\mathcal{GDG}$ solves the liveness of $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings. In the following proof, we consider that there exists an eventual missing edge while $\mathcal{GDG}$ is executed, otherwise, during the execution, the ring is a $\mathcal{RE}$ ring (we treat the case of $\mathcal{RE}$ rings in subsection 5.2).

Lemma 21. *$\mathcal{GDG}$ solves the liveness of $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings.*

Proof. By contradiction, assume that $\mathcal{GDG}$ does not solve the liveness of $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings. Since the execution of Rules $\mathbf{Term_1}$ and $\mathbf{Term_2}$ permits a robot to terminate its execution, by Lemma 18, this implies that there exists a $\mathcal{COT}$ ring such that, during the execution of $\mathcal{GDG}$, neither Rule $\mathbf{Term_1}$ nor Rule $\mathbf{Term_2}$ is executed. Consider the execution of $\mathcal{GDG}$ on that ring.

By Corollary 5, there exists a time t at which a $towerMin$ is formed. Note that $\mathcal{R}-2 \geq 2$ robots are involved in a $towerMin$. Once a $towerMin$ is formed the $\mathcal{R}-3$ $waitingWalker$ and the $minWaitingWalker$ involved in this $towerMin$ execute Rule $\mathbf{K_1}$. While executing this rule, the robot r with the maximum identifier among the $\mathcal{R}-2$ robots involved in this $towerMin$ becomes $headWalker$, the $minWaitingWalker$ becomes $minTailWalker$ and the other robots involved in this $towerMin$ become $tailWalker$. Note that, by Corollary 4, only r_{min} can be min, and therefore, since r_{min} is the robot with the minimum identifier among all the robots of the system and since at least 2 robots are involved in the $towerMin$, r_{min} cannot become $headWalker$. By Lemma 6 and by the rules of $\mathcal{GDG}$, only r can be $headWalker$ and only r_{min} can be $minTailWalker$ during the execution.

There is no rule in $\mathcal{GDG}$ permitting a $tailWalker$ or a $minTailWalker$ robot to become another kind of robot. A $tailWalker$ and a $minTailWalker$ can only execute Rules $\mathbf{T_3}$ and $\mathbf{W_1}$. By the rules of $\mathcal{GDG}$, the $minTailWalker$ and the $tailWalker$ execute the same movements at the same time starting from the same node, therefore, they are on a same node at each instant time. Hence, call *tail* the set of all of these robots.

A $headWalker$ can become a $leftWalker$. However, since we assume that the liveness of $\mathbb{G}_{EW}$ cannot be solved, then it is not possible for r to become a $leftWalker$. Indeed, a $headWalker$ can only execute Rules $\mathbf{T_2}$, $\mathbf{T_3}$ and $\mathbf{W_1}$. Note that, by the rules of $\mathcal{GDG}$, after the execution of Rule $\mathbf{K_1}$, the $headWalker$ and the *tail* both execute Rule $\mathbf{W_1}$. Therefore, since the $headWalker$ and the *tail* start the execution of Rule $\mathbf{W_1}$ on the same node at the same time, by the rules of $\mathcal{GDG}$, while the $headWalker$ is executing Rule $\mathbf{T_3}$ or Rule $\mathbf{W_1}$, if the *tail* is not on the same node as the $headWalker$, it is either executing Rule $\mathbf{W_1}$ or it is terminated. Moreover, by the same arguments, in the remaining of the execution, the $headWalker$ and the *tail* are either on a same node or the *tail* is on the left adjacent node (on the footprint of the dynamic ring) of the node where the $headWalker$ is located. Hence, if at a time t', the $headWalker$ executes Rule $\mathbf{T_2}$, and therefore becomes a $leftWalker$, then this implies that during time $t'-1$ it is executing either Rule $\mathbf{T_3}$ or Rule $\mathbf{W_1}$ while there is an adjacent left edge to its position and at time t' the *tail* is not on its node. Therefore, necessarily the *tail* is terminated, otherwise as explained the *tail* would have join the $headWalker$ on its node (Rule $\mathbf{W_1}$). Since only Rules $\mathbf{Term_1}$ and $\mathbf{Term_2}$ permit a robot to terminate its execution, by Lemma 18, this implies that the *tail* has executed Rule $\mathbf{Term_2}$, which leads to a contradiction with the fact that $\mathcal{GDG}$ does not solve the liveness of $\mathbb{G}_{EW}$.

Therefore, during the whole execution (after the execution of Rule $\mathbf{K_1}$), the $headWalker$, $tailWalker$ and $minTailWalker$ stay respectively $headWalker$, $tailWalker$ and $minTailWalker$ and can only execute Rule $\mathbf{W_1}$ until their variables $walkSteps$ reach n, and then they can only execute Rule $\mathbf{T_3}$.

Call r_1 and r_2 the two robots that are not involved in the $towerMin$ at time t. Since, by contradiction, neither Rule $\mathbf{Term_1}$ nor Rule $\mathbf{Term_2}$ are true, neither r_1 nor r_2 can meet the $headWalker$ or the *tail* while they (the $headWalker$ and the *tail*) are on a same node. Therefore, we assume that this event never happens.

By Lemma 17, at time t, $state_{r_1} \in \{righter, potentialMin, dumbSearcher, awareSearcher\}$ and $state_{r_2} \in \{dumbSearcher, awareSearcher\}$.

Let us first consider all the possible interactions between only r_1 and r_2 while $state_{r_1} \in \{righter, potentialMin, dumbSearcher, awareSearcher\}$ and $state_{r_2} \in \{dumbSearcher, awareSearcher\}$.

An isolated $potentialMin$ or a $potentialMin$ that is located only with a $dumbSearcher$ stays a $potentialMin$ and considers the *right* direction (Rule $\mathbf{M_8}$).

If a $potentialMin$ is located only with an $awareSearcher$, it becomes an $awareSearcher$ and it executes the function SEARCH (Rule $\mathbf{M_5}$).

An isolated *righter* stays a *righter* and considers the *right* direction (Rule $\mathbf{M_8}$).

If a *righter* is located only with a $dumbSearcher$ (resp. an $awareSearcher$), it becomes an $awareSearcher$ and executes the function SEARCH (Rule $\mathbf{M_7}$).

If a *dumbSearcher* is located only with a *righter*, it becomes an *awareSearcher* and executes the function SEARCH (Rule$\mathbf{M_9}$).

If a *dumbSearcher* is located only with a *potentialMin* it stays a *dumbSearcher* and executes the function SEARCH (Rule $\mathbf{M_{11}}$). In this case, while executing the function SEARCH, a *dumbSearcher* considers the $left$ direction, since it possesses a greater identifier than the one of the *potentialMin*. Indeed, only Rule $\mathbf{M_6}$ permits a robot to become *potentialMin* or *dumbSearcher*. This rule is executed when $\mathcal{R}-1$ *righter* are located on a same node. While executing Rule $\mathbf{M_6}$, among the $\mathcal{R}-1$ *righter*, the one with the minimum identifier becomes *potentialMin* while the others become *dumbSearcher*. By Observation 2, Rule $\mathbf{M_6}$ can be executed only once. Therefore, a *dumbSearcher* necessarily possesses an identifier greater than the one of the *potentialMin*.

An isolated *dumbSearcher* or a *dumbSearcher* located only with another *dumbSearcher* stays a *dumbSearcher* and executes the function SEARCH (Rule $\mathbf{M_{11}}$).

If a *dumbSearcher* is located only with an *awareSearcher*, it becomes an *awareSearcher* and it executes the function SEARCH (Rule $\mathbf{M_{10}}$).

An isolated *awareSearcher* or an *awareSearcher* located only with a *righter*, a *potentialMin*, a *dumbSearcher* or an *awareSearcher* stays an *awareSearcher* and executes the function SEARCH (Rule $\mathbf{M_{11}}$).

When r_1 and r_2 are on a same node without any other robot, executing the function SEARCH, since all the robots possess distinct identifiers, one considers the *right* direction, while the other one considers the $left$ direction.

While executing the function SEARCH at time i, a robot that is an isolated *dumbSearcher* or an isolated *awareSearcher* considers during the Move phase of time i the same direction it considers during the Move phase of time $i-1$. By Lemma 20, this direction cannot be equal to $\perp$.

By the previous movements described, note that, as long as r_1 and r_2 are not located with the *headWalker* or the *tail*, they are always such that $state_{r_1} \in \{righter, potentialMin, dumbSearcher, awareSearcher\}$ and $state_{r_2} \in \{dumbSearcher, awareSearcher\}$.

Now, consider the possible interactions between the *headWalker* and r_1 and/or r_2 when $state_{r_1} \in \{righter, potentialMin, dumbSearcher, awareSearcher\}$ and $state_{r_2} \in \{dumbSearcher, awareSearcher\}$.

If r_1 and/or r_2, as a *righter*, *potentialMin*, *dumbSearcher* or *awareSearcher* is on the same node as the *headWalker* such that there is no adjacent *right* edge to their location, then it executes Rule $\mathbf{M_3}$, hence it becomes an *awareSearcher* and stops to move.

($*$) If r_1 and/or r_2, as a *righter*, *potentialMin*, *dumbSearcher* or *awareSearcher* is on the same node as the *headWalker* such that there is an adjacent *right* edge to their location, then it executes Rule $\mathbf{M_2}$, hence it becomes an *awareSearcher* considering the right direction and therefore crosses the adjacent right edge to its node.

This implies that, as long as r_1 and r_2 are not located with the *tail* they are always such that $state_{r_1} \in \{righter, potentialMin, dumbSearcher, awareSearcher\}$ and $state_{r_2} \in \{dumbSearcher, awareSearcher\}$.

Finally, consider the possible interaction between the *tail* and r_1 and/or r_2. If r_1 and/or r_2, as a *righter*, *potentialMin*, *dumbSearcher* or *awareSearcher* is on the same node as the *minTailWalker*, then it executes Rule $\mathbf{M_4}$ and becomes a *tailWalker*. From this time, by the function BECOMETAILWALKER and the rules of $\mathcal{GDG}$, the robot belongs to the *tail*.

We assume that there exists an eventual missing edge. Call t" the time after the execution of Rule $\mathbf{K_1}$ and after the time when the eventual missing edge is missing forever. Consider the execution from t". Since Rule $\mathbf{K_1}$ is executed before time t", then there are *headWalker*, *tailWalker* and *minTailWalker* in the execution after time t" included.

Recall that, while executing Rules $\mathbf{W_1}$ and $\mathbf{T_3}$, the *headWalker* and the *tail* are either on a same node or on two adjacent nodes (the *tail* is on the adjacent left node on the footprint of the dynamic ring of the node where the *headWalker* is located).

Case 1: There is an eventual missing edge e between the node where the headWalker is located and the node where the tail is located.

As explained previously, since the *headWalker* and the *tail* are not on the same node, this necessarily implies that the *headWalker* either executes Rule $\mathbf{W_1}$ or Rule $\mathbf{T_3}$ at time t", and the *tail* executes Rule $\mathbf{W_1}$ at time t". Therefore, after time t", the *headWalker* does not move either because it waits for the *tail* to join it on its node (Rule $\mathbf{W_1}$), or because it executes the function STOPMOVING (Rule $\mathbf{T_3}$). Similarly, after time t", the *tail* does not move, since it tries to join the *headWalker* considering the right direction (Rule $\mathbf{W_1}$), but the edge is missing forever.

Since there is at most one eventual missing edge in a $\mathcal{COT}$ ring, all the edges, except e, are infinitely often present in the execution after time t". Considering the movements of the robots described previously, whatever the direction considered by r_1 and r_2 at time t" both of them succeed eventually to reach the node where the *tail* is located, making the liveness of $\mathbb{G}_{EW}$ solved.

Case 2: The eventual missing edge is not between the node where the headWalker is located and the node where the tail is located.

This implies that there exists a time from which the *headWalker* and the *tail* are located on a same node and do not move, either because they are executing Rule $\mathbf{T_3}$, or because they are executing Rule $\mathbf{W_1}$ but the adjacent *right* edge the *headWalker* tries to cross is the eventual missing edge. In the second case, by the movements of the robots described previously, we succeed to prove that, eventually at most one of the robots among r_1 and r_2 can be stuck on the extremity of the eventual missing edge where the *headWalker* and the *tail* are not located, and that at least one of them succeeds to reach the node where the *headWalker* and the *tail* are located, making the liveness of $\mathbb{G}_{EW}$ solved.

Consider now the first case. Call t_n the first time at which the *headWalker* and the *tail* are on a same node and both execute Rule $\mathbf{T_3}$. If r_1 and r_2 consider the same direction at time t_n, then by the movements of the robots described previously, whatever the place of the eventual missing edge, we succeed to prove that, eventually at most one of them can be stuck on one of the extremity of the eventual missing edge, and that at least one of them succeeds to reach the node where the *headWalker* and the *tail* are located, making the liveness of $\mathbb{G}_{EW}$ solved. Similarly, if the *headWalker* and the *tail* are located, at time t_n, on one of the extremity of the eventual missing edge, then, by the movements of the robots described previously, we succeed to prove that, eventually at most one of the robots among r_1 and r_2 can be stuck on the extremity of the eventual missing edge where the *headWalker* and the *tail* are not located, and that at least one of them succeeds to reach the node where the *headWalker* and the *tail* are located, making the liveness of $\mathbb{G}_{EW}$ solved.

Now consider the first case, when r_1 and r_2 consider opposed directions at time t_n and such that, at time t_n, the *headWalker* and the *tail* are not located on one of the extremity of the eventual missing edge. It is not possible for both r_1 and r_2 to be eventually stuck on two different extremities of the eventual missing edge. Indeed, if r_1 and r_2 consider two opposed directions at time t_n, this is because, between times t_i and t_n (with t_i the time at which the *headWalker* and the *tail* both execute Rule $\mathbf{W_1}$ for the first time), they are located on a same node (without any other robot on their node). We prove this by contradiction. Assume, by contradiction, that r_1 and r_2 are never located on a same node (without any other robot on their node) between times t_i and t_n. Consider the execution from time t_i until time t_n. Whatever the direction considered by r_1 (resp. r_2), it cannot be located with the *tail*, otherwise, since there is no eventual missing edge between the *headWalker* and the *tail* and by the movements of the robots described previously, Rule $\mathbf{Term_2}$ is eventually executed. Therefore, r_1 (resp. r_2) can only be located with the *headWalker*. When r_1 (resp. r_2) is located with the *headWalker*, it necessarily exists an adjacent right edge to their position before the adjacent left edge to their position appears, otherwise, the *tail* join them and Rule $\mathbf{Term_2}$ is executed. By $(*)$, after r_1 (resp. r_2) is on the same node as the *headWalker* while there is an adjacent *right* edge to their location, it becomes an *awareSearcher* considering the right direction. At time t_n, the *headWalker* and the *tail* execute Rule $\mathbf{T_3}$, therefore they succeed to execute Rule $\mathbf{W_1}$ until their variables *walkSteps* is equal to n. This implies that, if r_1 (resp. r_2) considers the left direction at time t_i, necessarily, since it cannot be located with r_2 (resp. r_1), by the movements of the robots described previously, it exists a time $t_{meet} \geq t_i$ at which the *headWalker* and the *tail* execute Rule $\mathbf{W_1}$ and either the *headWalker* or the *tail* is located with it. As explained previously, r_1 (resp. r_2) cannot be located with the *tail*, this implies that, at time t_{meet}, r_1 (resp. r_2) is located with the *headWalker*. Therefore, whatever the direction considered by r_1 (resp. r_2) at time t_i, if r_1 and r_2 are never located on a same node (without any other robot on their node) between times t_i and t_n, it necessarily considers the right direction at time t_n. Indeed, r_1 (resp. r_2) considers the right direction at time t_n either because it meets the *headWalker* that makes it consider the right direction or because at time t_i it considers the right direction and it is never located with the *headWalker* and, by the movements of the robots described previously, it has not change its direction between times t_i and time t_n. Hence, there is a contradiction with the fact that r_1 and r_2 consider opposite directions at time t_n. Therefore, r_1 and r_2 consider two opposite directions at time t_n because they are located on a same node (without any other robot on their node) between times t_i and t_n.

Consider the last time t_l between times t_i and t_n at which r_1 and r_2 are located on a same node (without any other robot on their node). At time t_l, since the two robots are located on a same node, by the movements of the robots described, during the Move phase of time t_l one considers the right direction while the other one considers the left direction. By assumption, between times $t_l + 1$ and t_n, r_1 and r_2 are not located on a same node. Moreover, as explained previously, between times $t_l + 1$ and t_n, neither r_1 nor r_2 can be located with the *tail*, otherwise Rule $\mathbf{Term_2}$ is eventually executed. Besides, between times $t_l + 1$ and t_n the robot that considers the left direction during the Move phase of time t_l cannot be located with the *headWalker*, otherwise, as noted previously, it considers the right direction at time t_n. Similarly, it is not possible for the robot that considers the *right* direction during the Move phase of time t_l to be located with the *headWalker* between times $t_l + 1$ and t_n, otherwise, by the movements of

the robots described previously, this necessarily implies that either it is also located on the same node as the *tail* and therefore the liveness of $\mathbb{G}_{EW}$ is solved or r_1 and r_2 are on a same node and therefore the robot that considers the left direction during the Move phase of time t_l is located with the $headWalker$. Therefore, from time t_l+1 to time t_n, r_1 and r_2 are isolated, hence, by the movements of the robots, they consider the same respective directions from the Move phase of time t_l to time t_n.

Assume, without lost of generality, that this is r_1 that considers the right direction from the Move phase of time t_l to time t_n. Call v_1 (resp. v_2) the node on which r_1 (resp. r_2) is located at time t_n. The explanations of the previous paragraph imply that $v_1 \neq v_2$, and that, at time t_n, the node where the $headWalker$ and the *tail* are located is in $Seg(v_1, v_2)$. Therefore, since r_1 (resp. r_2) considers the right (resp. the left) direction at time t_n, by the movements of the robots and since it exists only one eventual missing edge, this is not possible for these two robots to be eventually stuck on each of the extremities of the eventual missing edge. Hence, at least one succeeds to reach the node where the $headWalker$ and the *tail* are located, making the liveness of $\mathbb{G}_{EW}$ solved.

□

By Lemmas 19 and 21, we can deduce the following theorem which proves the correctness of Phases `W` and `T`.

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

5.2 What about $\mathcal{GDG}$ executed in $\mathcal{AC}$, $\mathcal{RE}$, $\mathcal{BRE}$ and $\mathcal{ST}$ rings?

In the previous subsection we prove that $\mathcal{GDG}$ solves $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings. In this subsection, we consider $\mathcal{AC}$, $\mathcal{RE}$, $\mathcal{BRE}$ and $\mathcal{ST}$ rings. For each of these classes of dynamic rings, we give the version of gathering $\mathcal{GDG}$ solves in it.

First, we consider the case of $\mathcal{AC}$ rings. In the following theorem, we prove that $\mathcal{GDG}$ solves $\mathbb{G}_W$ in $\mathcal{AC}$ rings.

Theorem 3. *$\mathcal{GDG}$ solves $\mathbb{G}_W$ in $\mathcal{AC}$ rings.*

Proof. By Corollary 2, $\mathcal{GDG}$ solves $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings, since $\mathcal{AC} \subset \mathcal{COT}$, this implies that $\mathcal{GDG}$ also solves $\mathbb{G}_{EW}$ in $\mathcal{AC}$ rings. Therefore, to prove that $\mathcal{GDG}$ solves $\mathbb{G}_W$ in $\mathcal{AC}$ rings, it stays to prove that each phase of $\mathcal{GDG}$ is bounded.

Phase `M`: By Corollary 4, only r_{min} becomes min in finite time. By the rules of $\mathcal{GDG}$, when r_{min} becomes min, it is first $minWaitingWalker$ before being $minTailWalker$ (since only a $minWaitingWalker$ can become a $minTailWalker$ while executing Rule $\mathbf{K_1}$). Therefore, since only Rule $\mathbf{M_1}$ permits a robot to become $minWaitingWalker$, by the predicate $MinDiscovery()$ of this rule, r_{min} becomes min either because it moves during $4 * n * id_{r_{min}}$ steps in the right direction or because it meets a robot that permits it to deduce that it is min. In this last case, note that, either r_{min} is $potentialMin$, or r_{min} meets a $potentialMin$ or a $dumbSearcher$ or a robot whose variable $idMin$ is different from $\perp$. Therefore, in this last case, either r_{min} possesses a variable $idPotentialMin$ different from $\perp$, or r_{min} meets a robot r such that $idPotentialMin_r$ is different from $\perp$ (since a $potentialMin$ and a $dumbSearcher$ have their variable $idPotentialMin$ different from $\perp$ (Rule $\mathbf{M_6}$) and since, while executing $\mathcal{GDG}$, each time the variable $idMin$ of a robot is set with a variable different from $\perp$, this is also the case for its variable $idPotentialMin$).

Taking back the arguments used in the proof of Lemma 5, let us consider the following cases.

Case 1.1: Rule $\mathbf{M_6}$ is never executed.

By the rules of $\mathcal{GDG}$, this implies that, before the time when r_{min} is min, there are only $righter$ in the execution. First, this implies that r_{min} becomes min because it moves during $4 * n * id_{r_{min}}$ steps in the $right$ direction (since $righter$ robots have their variables $idPotentialMin$ equal to $\perp$). Second, in this context, as long as r_{min} is not min, all the $righter$ always consider the $right$ direction (Rule $\mathbf{M_8}$). This implies that, as long as r_{min} is not min, each time a robot wants to move in the right direction it can be stuck during at most n rounds, otherwise, since in an $\mathcal{AC}$ ring at most one edge can be missing at each instant time, Rule $\mathbf{M_6}$ is executed. Therefore in case 1.1 r_{min} becomes min in at most $4 * id_{r_{min}} * n * n$ rounds.

Now let consider the case where Rule $\mathbf{M_6}$ is executed at a time t. In the following, we consider the execution from time t. After time t, while it is not yet min, if r_{min} is stuck more than $4 * n$ consecutive rounds on a same node then it becomes min. We prove this considering the two following cases. In each of these cases we assume that r_{min} is not yet min and that it is stuck more than n rounds on a same node.

Case 1.2: Rule $\mathbf{M_6}$ is executed but $\mathbf{r_{min}}$ is not among the $\mathcal{R}-1$ righter robots that execute it.

Taking back the arguments of the proof of Lemma 5, we know that Rule $\mathbf{M_6}$ can be executed only once, and that the robots that execute this rule can never be *righter* anymore. Moreover, since r_{min} does not execute Rule $\mathbf{M_6}$, since, by Corollary 4, r_{min} necessarily becomes *min*, since, by Lemma 9, only *righter* and *potentialMin* can be *min*, and since only Rule $\mathbf{M_6}$ permits robots to become *potentialMin*, before becoming *min*, r_{min} is a *righter*. By the proof of Lemma 5, as long as r_{min} is not *min* it cannot exist *awareSearcher*. Hence, by the rules of $\mathcal{GDG}$, as long as r_{min} is not *min*, there are only one *righter*, one *potentialMin* and $\mathcal{R}-2$ *dumbSearcher* in the execution. Therefore, by the rules of $\mathcal{GDG}$, the *potentialMin* is *potentialMin* at least until r_{min} becomes *min*. Hence, the *potentialMin* executes Rule $\mathbf{M_8}$ and thus considers the right direction at least until r_{min} becomes *min*. We have assumed that, while it is not yet *min*, r_{min} is stuck more than $4*n$ consecutive rounds on a same node. Since r_{min} is a *righter* before being *min*, it is stuck because the adjacent right edge to its position is missing (Rule $\mathbf{M_8}$). Therefore, since in an $\mathcal{AC}$ ring of size n at least $n-1$ edges are present at each instant time, either the *potentialMin* (or a *dumbSearcher*) meets r_{min} in at most n rounds. When r_{min} meets a *potentialMin* (or a *dumbSearcher*), it becomes *min* by definition of the predicate *MinDiscovery*() in Rule $\mathbf{M_1}$. Therefore, if it is stuck more than $4*n$ rounds, r_{min} becomes *min* in at most n rounds.

Case 1.3: Rule $\mathbf{M_6}$ is executed and $\mathbf{r_{min}}$ is among the $\mathcal{R}-1$ righter robots that execute it.

In this case, by Rule $\mathbf{M_6}$, r_{min} becomes *potentialMin*. By Observations 2 and 1, by Corollary 4 and by Lemma 9 r_{min} is *potentialMin* until it becomes *min*. Therefore, r_{min}, while it is not yet *min*, can be stuck only because the adjacent right edge to its position is missing (Rule $\mathbf{M_8}$).

First, consider that at the time when r_{min}, as a *potentialMin*, is stuck more than $4*n$ rounds, there does not exist *righter* in the execution. By Observation 2, there is no more *righter* in the execution. However, at the time when Rule $\mathbf{M_6}$ is executed, the robot r that is not among the robots that execute this rule is a *righter*. Therefore, necessarily r, as a *righter*, meets at least one *dumbSearcher* at a time t'. Indeed, it cannot meet the *potentialMin*, otherwise r_{min} is *min* (Rule $\mathbf{M_1}$), and thus it is not anymore *potentialMin* at the time at which it is stuck. Moreover, r cannot be isolated forever after time t, otherwise it stays a *righter* (Rule $\mathbf{M_8}$). Hence, at time t', r becomes an *awareSearcher* (Rule $\mathbf{M_7}$). Consider an *awareSearcher* r_a of the execution. By Lemma 4, r_a cannot consider the $\perp$ direction. Moreover, by the rules of $\mathcal{GDG}$, as long as there is no *min*, an *awareSearcher* executes the function SEARCH (rule $\mathbf{M_{11}}$). Besides, by the proof of Lemma 5 if a robot is not isolated and executes the function SEARCH, then all the robots of its node are or become *awareSearcher* and execute the function SEARCH. While executing the function SEARCH, an isolated robot does not change its direction. When a robot executes the function SEARCH while there are multiple robots on its node, if it possesses the maximum identifier among the robots of its node, it considers the left direction, otherwise it considers the right direction. In an $\mathcal{AC}$ ring of size n, at least $n-1$ edges are present at each instant time. Therefore, if r_a considers the *right* direction, either it, as an *awareSearcher* or a robot that is or becomes an *awareSearcher* is located, in at most n rounds, on the node where r_{min}, as a *potentialMin*, is stuck. In the case where r_a considers the $left$ direction then, by the same arguments, in at most $4*n$ rounds an *awareSearcher* is located on the node where r_{min}, as a *potentialMin*, is stuck. Indeed, at most n rounds are needed for an *awareSearcher*to reach the extremity of the missing edge where r_{min} is not located. Then, at most $2*n$ other rounds are needed for a *dumbSearcher* (execution of the function SEARCH, rule $\mathbf{M_{11}}$) or an *awareSearcher* to reach also this node. These $2*n$ rounds are especially needed for a *dumbSearcher* that may take n rounds (considering the left direction) to reach the node where r_{min} is stuck and then again n rounds (considering the right direction) to reach the other extremity of the missing edge. From this time there is in the execution an *awareSearcher* that considers the *right* direction. Finally, at most n supplementary rounds are needed for an *awareSearcher* to reach the node where r_{min}, as a *potentialMin*, is stuck. Note that $\mathcal{R}>4$, and there are $\mathcal{R}-1$ *dumbSearcher/awareSearcher* in the execution as long as r_{min} is not *min*. Therefore, the previous scenario can effectively happen. When r_{min} meets an *awareSearcher*, it becomes *min* by definition of the predicate *MinDiscovery*() of rule $\mathbf{M_1}$. Therefore, r_{min} becomes *min* in at most $4*n$ rounds if it is stuck more than $4*n$ rounds.

Second, consider that at the time when r_{min}, as a *potentialMin*, is stuck more than $4*n$ rounds, there exists a *righter*. In this case, since an isolated *righter* considers the *right* direction (Rule $\mathbf{M_8}$), and by the arguments of the previous paragraph, either a *righter* or a robot that is an *awareSearcher* or that becomes an *awareSearcher*(Rules $\mathbf{M_7}$, $\mathbf{M_9}$ or $\mathbf{M_{10}}$) meets r_{min} in at most n rounds. When r_{min} meets a *righter* or an *awareSearcher*, it becomes *min* by definition of the predicate *MinDiscovery*() of Rule $\mathbf{M_1}$. Therefore, r_{min} becomes *min* in at most n rounds if it is stuck more

than $4 * n$ rounds.

Now, we give the worst number of rounds needed for r_{min} to become min, in the case where there exists a time t at which Rule $\mathbf{M_6}$ is executed. By Case 1.1, before time t, r_{min}, while it is not yet min, can be stuck at most n rounds each time it moves from one step in the $right$ direction. Similarly, by the two previous cases (Case 1.2 and 1.3), after time t, r_{min}, while it is not yet min, can be stuck at most $4*n$ rounds each time it moves from one step in the $right$ direction. Let nb be the number of steps in the right direction moved by r_{min} before time t. As proved previously, r_{min} is either a $righter$ or a $potentialMin$ before being min. By Lemma 2, this implies that before being min, r_{min} always considers the right direction. Therefore, by the predicate $MinDiscovery()$ of Rule $\mathbf{M_1}$, in at most $nb*n+((4*id_{r_{min}}*n)-nb)*4*n$ rounds, r_{min} becomes min because it moves during $4*id_{r_{min}}*n$ steps in the right direction. This function is maximal when $nb=0$, therefore in at most $16*id_{r_{min}}*n^2$ rounds r_{min} becomes min because it moves during $4*id_{r_{min}}*n$ steps in the right direction. Now consider the case where r_{min} becomes min because it meets a robot that permits it to deduce that it is min. Once r_{min} is stuck more than $4*n$ rounds after time t, we have seen that it becomes min. Since we consider the worst case such that r_{min} does not become min because it moves during $4*id_{r_{min}}*n$ steps in the right direction, this implies that in at most $(4*id_{r_{min}}*n-1)*4*n+4*n$ rounds r_{min} becomes min. Therefore, whatever the situation, Phase M is bounded.

Now we consider Phase K of $\mathcal{GDG}$. In this phase r_{min} is min and waits for a $towerMin$ to be formed. We take back the arguments used in the proofs of Lemmas 15 and 16 to prove that this phase is bounded.

Phase K: Case 2.1: There is a potentialMin in the execution.

For this case we take back the arguments of the proof of Lemma 15.

If before being min, r_{min} is a $righter$, then all the robots that are not located on node u are $potentialMin$, $dumbSearcher$, and $awareSearcher$. As long as it is not on node u, a $potentialMin$ either executes Rule $\mathbf{M_8}$, or it becomes an $awareSearcher$ (Rule $\mathbf{M_5}$). While executing Rule $\mathbf{M_8}$, a $potentialMin$ stays a $potentialMin$ and has the same behavior as if it was executing the function SEARCH. Moreover, as long as they are not on node u, $dumbSearcher$ and $awareSearcher$ robots stay either $dumbSearcher$ or $awareSearcher$ and execute the function SEARCH. Therefore, by definition of the function SEARCH (refer to Phase M case 1.3 of this proof) and by Lemma 13, at most $3*n$ rounds are needed (in $\mathcal{AC}$ rings) for a robot r such that $state_r \in \{potentialMin, dumbSearcher, awareSearcher\}$ to be located on node u. Indeed, these $3*n$ rounds are needed especially when a $potentialMin$, $dumbSearcher$ or $awareSearcher$ moves in one direction during n steps and then is stuck on the adjacent node of u, then n steps are needed for a robot of this kind to be also located on this node and thus to consider the opposite direction, then in at most n additional steps a robot of this kind is located on u. By Rule $\mathbf{K_3}$, this implies that at most $3*n$ rounds are necessary for a supplementary $waitingWalker$ to be located on node u. Therefore, at most $(\mathcal{R}-3)*3*n$ rounds are needed for a $towerMin$ to be formed.

Now consider the case where before being min, r_{min} is a $potentialMin$.

In this case among the robots that are not on node u, there are $dumbSearcher$, $awareSearcher$ and at most one $righter$.

For all the cases of Case 2.1 of the proof of Lemma 15, at most $(\mathcal{R}-4)*3*n+3*n$ rounds are needed for a $towerMin$ to be formed. Indeed, at most $(\mathcal{R}-4)*3*n$ rounds are needed for $\mathcal{R}-4$ $waitingWalker$ to be located on u (for the same reasons as the one explained in the previous paragraph). Then among the robots that are not on node u, it exists at most one $righter$, and 2 robots that are either $dumbSearcher$ or$awareSearcher$. At most n rounds are needed for the $righter$to be stuck on the node called v in the proof of Lemma 15, and then at most n rounds are needed for a $dumbSearcher$ or an $awareSearcher$ to be also located on node v (and thus, by Rule $\mathbf{M_7}$, for all the robots that are not on node u to be either $dumbSearcher$ or $awareSearcher$), and then at most n additional rounds are needed for one of the robot to reach node u.

If we consider Case 2.2 of the proof of Lemma 15, similarly as in the previous case, at most $(\mathcal{R}-4)*3*n+3*n$ rounds are needed for Rule $\mathbf{Term_2}$ to be executed.

Case 2.2: There is no potentialMin in the execution.

For this case we take back the arguments of the proof of Lemma 16.

Just after r_{min} becomes min, it takes at most $n*n$ rounds for a robot r to join the node where r_{min} is located. Indeed, as long as no robot is on node u with r_{min}, as a $minWaitingWalker$, all the robots except r_{min} are $righter$. By the same arguments than the one used in Phase M Case 1.1 of this proof, a $righter$ cannot be stuck more than n rounds on the same node, otherwise Rule $\mathbf{M_6}$ is executed, which is a contradiction with the fact that there is no $potentialMin$. Moreover, a $righter$ can move from at most n steps in the right direction to reach u.

Once r is on node u an adjacent right edge to u is present in at most n rounds, otherwise Rule **Term$_1$** is executed. Therefore, once r is on node u, in at most n rounds it becomes an *awareSearcher*. From this time, either it is possible for all the *righter* to become *awareSearcher* or it exists at least one *righter* that is stuck on node u. In the first case at most $2*n$ rounds are needed for all the *righter* to become *awareSearcher* (either because an *awareSearcher* meets them, or because they are located on u such that there is an adjacent right edge to u). By the arguments above, we know that if all the robots that are not located on node u are *awareSearcher*, and if there are more than 3 such robots, then in a most $3*n$ rounds one robot of this kind reaches node u. Therefore, for $\mathcal{R}-3$ *waitingWalker* to be located on node u, with r_{min}, at most $(\mathcal{R}-3)*3*n$ supplementary rounds are needed. In the second case, at most $2*n$ rounds are needed for some of the *righter* to reach node u (and to be stuck on this node). Since the robots that are not on node u are *awareSearcher* and since at least one *righter* is stuck on node u, by the same arguments as above, at most $(\mathcal{R}-3)*3*n$ additional rounds are needed for Rule **Term$_2$** to be executed.

Therefore Phase K is bounded.

Now we consider Phase W of $\mathcal{GDG}$. In this purpose we take back the arguments used in the proof of Lemma 21.

Phase W: Here we consider the worst execution in terms of times. Therefore, we consider that Rules **Term$_1$** and **Term$_2$** are executing at the very last moment. The robots r_1 and r_2 that are not involved in T at time t_{tower} are such that $state_{r_1} \in \{righter, potentialMin, dumbSearcher, awareSearcher\}$ and $state_{r_2} \in \{dumbSearcher, awareSearcher\}$. Therefore, as explained previously, each time the *headWalker*, or the *minTailWalker / tailWalker* robots move from one steps in the right direction, they can be stuck at most during $3*n$ rounds, otherwise either Rule **Term$_1$** or Rule **Term$_2$** is executed. Indeed, this is especially the case when the *headWalker* and the *minTailWalker / tailWalker* are stuck on the same node. In fact, it takes at most n rounds for r_1 to be stuck on the other extremity the missing edge. At most n supplementary rounds are needed for r_2 to reach the node where r_1 is stuck (and therefore for one robot to change its direction), and then n other rounds are needed for one of these robots to reach the node where the *headWalker* and the *minTailWalker / tailWalker* are stuck (and thus for Rule **Term$_2$** to be executed). Therefore, Phase W is achieved in at most $2*n*(3*n)$ rounds since the *headWalker* and the *minTailWalker / tailWalker* robots have to move alternatively during n steps to complete Phase W. In other words, Phase W is bounded.

Now we consider Phase T of $\mathcal{GDG}$. In this purpose we take back the arguments used in the proof of Lemma 21.

phase T: Using similar arguments as the one used in Phase W, once the *headWalker* and the *minTailWalker / tailWalker* stop to move forever, if they are located on a same node, at most $3*n$ rounds are necessary for Rule **Term$_2$** to be executed. In the case where the *headWalker* and the *minTailWalker / tailWalker* stop to move forever, if they are located on different nodes, at most $2*n+2*n$ rounds are necessary for Rule **Term$_2$** to be executed. Indeed, at most $2*n$ rounds are necessary for each of the two robots that are not involved in T at time t_{tower} to be located on the node where the *minTailWalker /tailWalker* is located. This is true whatever the interactions between r_1 and r_2 and whatever the interactions between r_1 (resp. r_2) and the *headWalker* since in an $\mathcal{AC}$ ring there is at most one edge missing at each instant time (and in this precise case the missing edge is between the node where the *headWalker* is located and the node where the *minTailWalker / tailWalker* are located).

In conclusion each of the four phases of algorithm $\mathcal{GDG}$ are bounded when executed in an $\mathcal{AC}$ ring, therefore $\mathcal{GDG}$ solves $\mathbb{G}_W$ in $\mathcal{AC}$ rings. □

Now, we consider the case of $\mathcal{RE}$ rings. In the following theorem, we prove that $\mathcal{GDG}$ solves $\mathbb{G}_E$ in $\mathcal{RE}$ rings.

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

Proof. By Corollary 2, $\mathcal{GDG}$ solves $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings, therefore it solves the safety and the liveness of $\mathbb{G}_{EW}$ in $\mathcal{COT}$ rings. Since $\mathcal{RE} \subset \mathcal{COT}$, $\mathcal{GDG}$ also solves the safety and the liveness of $\mathbb{G}_{EW}$ in $\mathcal{RE}$ rings. This implies that all robots that terminate their execution terminate it on the same node and it exists a time at which at least $\mathcal{R}-1$ robots terminate their execution. Call t the first time at which at least $\mathcal{R}-1$ robots terminate their execution.

By contradiction, assume that $\mathcal{GDG}$ does not solve $\mathbb{G}_E$ in $\mathcal{RE}$ rings, this implies that it exists a robot r that never terminates its execution.

Call *towerTermination* the $\mathcal{R}-1$ robots that, at time t, are located on a same node and are terminated. While executing $\mathcal{GDG}$, the only way for a robot to terminate its execution is to execute either Rule **Term$_1$** or

Rule $\mathbf{Term_2}$. By Lemma 18, for a *towerTermination* to be formed at time t, Rule $\mathbf{Term_2}$ has to be executed at this time.

$(*)$ By the predicate of Rule $\mathbf{Term_2}$, r_{min} belongs to the *towerTermination*. By Lemma 18, all the robots that are located on the same node as r_{min} at time t belong to the *towerTermination*.

Call w the node where the *towerTermination* is located at time t.

Note that r cannot be located on node w after time t included, otherwise it executes Rule $\mathbf{Term_1}$ and the lemma is proved.

Since r_{min} belongs to the *towerTermination*, and since by Corollary 4, only r_{min} can be *minWaitingWalker* or a *minTailWalker*, r is neither *minWaitingWalker* nor *minTailWalker*.

At time t, r cannot be a *tailWalker*. Indeed, to become a *tailWalker*, a robot must either execute Rule $\mathbf{K_1}$ or Rule $\mathbf{M_4}$. To execute Rule $\mathbf{K_1}$ a robot must be a *waitingWalker*. By Lemma 7, all *waitingWalker* are located on the same node as the *minWaitingWalker*. Moreover, when a *waitingWalker* executes Rule $\mathbf{K_1}$, by the predicate *AllButTwoWaitingWalker*(), the *minWaitingWalker* also executes this rule becoming a *minTailWalker*. Then by the rules of $\mathcal{GDG}$, the robot that becomes *tailWalker* while executing Rule $\mathbf{K_1}$ and the *minTailWalker* execute the same movements (refer to Rules $\mathbf{W_1}$ and $\mathbf{T_3}$), and therefore are always on a same node. Besides, to execute Rule $\mathbf{M_4}$ a robot must be located on the same node as the *minWaitingWalker* (refer to the predicate *NotWalkerWithTailWalker*(r')). Then, thanks to the function BecomeTailWalker and by the rules of $\mathcal{GDG}$, the robot that becomes *tailWalker* while executing Rule $\mathbf{M_4}$ cannot be on a node different from the one where the *minTailWalker* is located (refer to Rules $\mathbf{W_1}$ and $\mathbf{T_3}$). Therefore, by $(*)$, r cannot be a *tailWalker* at time t, otherwise, at time t, it is on the same node as the *minTailWalker* (thus, by Corollary 4, it is on the same node as r_{min}) and hence it terminates its execution.

At time t, r cannot be a *waitingWalker* robot. Indeed by the rules of $\mathcal{GDG}$ and the previous remarks, it cannot exists *waitingWalker* if there is no *minWaitingWalker* in the execution, and by Lemma 7 all the *waitingWalker* and *minWaitingWalker* are located on a same node. Therefore, by $(*)$, r cannot be a *waitingWalker* at time t, otherwise, at time t, it is on the same node as the *minWaitingWalker* (thus, by Corollary 4, it is on the same node as r_{min}) and hence it terminates its execution.

Therefore, at time t, r can be either a *righter*, a *potentialMin*, a *dumbSearcher*, an *awareSearcher*, a *headWalker* or a *leftWalker* robot.

As long as r is not on node w, it is isolated.

An isolated *righter* or an isolated *potentialMin* only executes Rule $\mathbf{M_8}$. While executing this rule, a robot considers the *right* direction and stays a *righter* or a *potentialMin*. Since all the edges are infinitely often present, such a robot is infinitely often able to move in the *right* direction until reaching the node w.

An isolated *dumbSearcher* or an isolated *awareSearcher* only executes Rule $\mathbf{M_{11}}$. While executing this rule, an isolated robot stays a *dumbSearcher* or an *awareSearcher*, and considers the direction it considers during the previous Move phase. By Lemma 20, this direction cannot be equal to $\perp$. Therefore, an isolated *dumbSearcher* or an isolated *awareSearcher* always considers the same direction d (either *right* or *left*). Since all the edges are infinitely often present, such a robot is infinitely often able to move in the direction d until reaching the node w.

Now assume that, at time t, r is a *leftWalker*. A *leftWalker* only executes Rule $\mathbf{T_1}$. While executing this rule, a robot considers the *left* direction and stays a *leftWalker*. Since all the edges are infinitely often present, such a robot is infinitely often able to move in the *left* direction until reaching the node w.

Now assume that, at time t, r is a *headWalker*. A *headWalker* can execute either Rule $\mathbf{T_2}$ or Rule $\mathbf{T_3}$ or Rule $\mathbf{W_1}$. While executing Rule $\mathbf{T_2}$, a *headWalker* becomes a *leftWalker*, then, by the previous paragraph, r reaches the node w in finite time. Consider now the cases where, at time t, r executes either Rule $\mathbf{T_3}$ or Rule $\mathbf{W_1}$. In these cases, after time t, it necessarily exists a time at which r executes Rule $\mathbf{T_2}$. Assume, by contradiction, that this is not true. The only way for a robot to become a *headWalker* is to execute Rule $\mathbf{K_1}$. Rule $\mathbf{K_1}$ is executed when $\mathcal{R}-2$ robots are located on a same node. While executing this rule, a robot sets its variable *walkerMate* with the identifiers of the robots that are located on its node. Only Rule $\mathbf{K_1}$ permits a robot to update its variable *walkerMate*. Note that, since $\mathcal{R}-2 \geq 2$, the variable *walkerMate* of r, after time t, contains at least one identifier i different from the identifier of r. The robot of identifier i necessarily belongs to the *towerTermination*, since only r does not terminate. (1) Hence, at time t, the robot of identifier i is terminated on node w, thus it does not move, and therefore, after time t, r is never on the same node as i. (2) All the edges are infinitely often present. While executing Rule $\mathbf{T_3}$ at time t, r considers the $\perp$ direction and does not update its other variables. (3) Hence, by the rules of $\mathcal{GDG}$, since r cannot execute Rule $\mathbf{T_2}$, after time t, r can only execute Rule $\mathbf{T_3}$, and therefore only considers the $\perp$ direction. Hence, necessarily by (1), (2) and (3), this implies that, after time t, it exists a time at which the predicate *HeadWalkerWithoutWalkerMate*() is true, thus at this time Rule $\mathbf{T_2}$ is executed. Similarly, if at time t, r executes Rule $\mathbf{W_1}$, since r can never be located on the same node as i, while executing Rule $\mathbf{W_1}$, it considers the $\perp$ direction and does not update its other variables. (4) Hence, by the rules of $\mathcal{GDG}$, since r cannot execute Rule $\mathbf{T_2}$, after time t, r can only execute Rule $\mathbf{W_1}$, and therefore always considers the $\perp$ direction. Thus, by (1), (2) and (4), necessarily, after time t, it exists a time at which the predicate *HeadWalkerWithoutWalkerMate*() is true, hence at this time

Rule $\mathbf{T_2}$ is executed. Therefore, even in the cases where, at time t, r executes either Rule $\mathbf{T_3}$ or Rule $\mathbf{W_1}$, it exists a time greater than t at which r becomes a $leftWalker$ and hence, by the previous paragraph, r succeeds to reach the node w in finite time.

Therefore whatever the kind of robot r is, it is always able to reach the node w. Once r reaches the node w it executes Rule $\mathbf{Term_1}$ making $\mathbb{G}_E$ solved. □

Now, we consider the case of $\mathcal{BRE}$ rings. We prove, in Theorem 5, that $\mathcal{GDG}$ solves $\mathbb{G}$ in $\mathcal{BRE}$ rings. To prove this, we first need to prove the following lemma that it useful to bound Phase K of $\mathcal{GDG}$ in $\mathcal{BRE}$ rings.

Lemma 22. *If the ring is a $\mathcal{BRE}$ ring and if there is no $towerMin$ in the execution but there exists at a time t at least 3 robots such that they are either $potentialMin$, $dumbSearcher$ or $awareSearcher$, then at least a $potentialMin$, a $dumbSearcher$ or an $awareSearcher$ reaches the node u between time t and time $t + n * \delta$ included, with $\delta \geq 1$.*

Proof. We prove this lemma using the arguments of the proof of Lemma 14 and the fact that in a $\mathcal{BRE}$ ring each edge appears at least once every δ units of time. □

Theorem 5. *$\mathcal{GDG}$ solves $\mathbb{G}$ in $\mathcal{BRE}$ rings.*

Proof. By Lemma 4, $\mathcal{GDG}$ solves $\mathbb{G}_E$ in $\mathcal{RE}$ rings. Therefore, since $\mathcal{BRE} \subset \mathcal{RE}$, then $\mathcal{GDG}$ also solves $\mathbb{G}_E$ in $\mathcal{BRE}$ rings. We want to prove that $\mathcal{GDG}$ solves $\mathbb{G}$ in $\mathcal{BRE}$rings. Therefore, we have to prove that each phase of the algorithm is bounded.

Phase M: By Corollary 4, we know that only r_{min} becomes min in finite time. By Lemma 9, before being min, r_{min} is either a $righter$ or a $potentialMin$ robot. By Lemma 2, if, at a time t, a robot is a $righter$ or a $potentialMin$ robot, then it considers the $right$ direction from the beginning of the execution until the Look phase of time t. Since initially all the robots are $righter$, and since, by the rules of $\mathcal{GDG}$, only $righter$ can become $potentialMin$ (refer to Rule $\mathbf{M_6}$), then by Observations 2 and 1, a robot that is a $righter$ (resp. $potentialMin$) is a $righter$ (resp. is either a $righter$ or a $potentialMin$) since the beginning of the execution. Besides, each time r_{min}, as a $righter$ or as a $potentialMin$, crosses an edge in the right direction, it increases its variable $rightSteps$ of one (refer to Rules $\mathbf{M_8}$ and $\mathbf{M_6}$). Therefore, since each edge of the footprint of a $\mathcal{BRE}$ring is present at least once every δ units of time, by definition of min and of the predicate $MinDiscovery()$ of Rule $\mathbf{M_1}$, r_{min} becomes min in at most $4 * n * id_{r_{min}} * \delta$ rounds. Hence, Phase M is bounded.

Phase K: Now, consider the execution when r_{min} just becomes min. Therefore, we consider the execution once r_{min} is $minWaitingWalker$. By Corollary 5, we know that in finite time a $towerMin$ is formed. By Lemma 6, there is only one $towerMin$ in the whole execution. Therefore, before a $towerMin$ is formed, by the rules of $\mathcal{GDG}$ and since initially all the robots are $righter$, there are only $righter$, $potentialMin$, $dumbSearcher$, $awareSearcher$, $minWaitingWalker$ and $waitingWalker$ robots. By Lemma 7, we know that all the $minWaitingWalker$ and $waitingWalker$ robots are located on a same node and do not move. By Rule $\mathbf{K_3}$, if a $potentialMin$, a $dumbSearcher$ or an $awareSearcher$ is located on the same node as a $minWaitingWalker$, it becomes $waitingWalker$ $(*)$. If there is no more $righter$ robot in the execution, we use Lemma 22 and $(*)$ multiple times to prove that it takes at most $n * \delta * (\mathcal{R} - 3)$ rounds for a $towerMin$ to be formed. To prove that Phase K is bounded, we hence have to prove that the number of rounds that are necessary to stop to have $righter$ in the execution is bounded.

If a $righter$ is located on the same node as the $minWaitingWalker$ while there is an adjacent right edge to its location, then by Rule $\mathbf{K_4}$, the $righter$ becomes an $awareSearcher$ and moves on the right. If a $righter$ is located only with $\mathcal{R}-2$ other $righter$, they all execute Rule $\mathbf{M_6}$, hence one becomes $potentialMin$ while the others become $dumbSearcher$. If a $righter$ is located either with a $dumbSearcher$ or with an $awareSearcher$, then it becomes an $awareSearcher$ (Rule $\mathbf{M_7}$). Note that, by Lemma 10, since we consider the execution once r_{min} is min, it cannot exist a $righter$ and a $potentialMin$ in the execution. Therefore, a $righter$ cannot meet a $potentialMin$. In all the other cases, (a $righter$ that is isolated, a $righter$ that is only with other $righter$ on its node such that $|NodeMate()| < \mathcal{R}-2$, and a $righter$ that is located on the same node as the $minWaitingWalker$ while there is no adjacent right edge to its location) a $righter$ stays a $righter$ and considers the $right$ direction (Rule $\mathbf{M_8}$). Therefore, by Observation 2 and since each edge of the footprint of a $\mathcal{BRE}$ring is present at least once every δ units of time, it takes at most $n * \delta$ rounds in order to stop having $righter$ robots in the execution. Indeed, even if a $righter$ does not execute Rule $\mathbf{M_7}$ or Rule $\mathbf{M_6}$, at most $n * \delta$ rounds are needed for it to be located on the node where the $minWaitingWalker$ is located while there is an adjacent right edge to its position. Hence, Phase K is bounded.

Once a $towerMin$ is present in the execution, the robots forming this $towerMin$ execute Rule $\mathbf{K_1}$. While executing this rule, the robot r with the maximum identifier among the robots involved in this $towerMin$ becomes $headWalker$ while the $minWaitingWalker$ becomes $minTailWalker$ and the other robots involved in this $towerMin$ become $tailWalker$. Note that, by Corollary 4, only r_{min} can be min, and therefore, since r_{min} is the robot with the minimum identifier among all the robots of the system and since at least 2 robots are involved in a $towerMin$, r_{min} cannot become $headWalker$. By Lemma 6 and by the rules of $\mathcal{GDG}$, only r can be $headWalker$ during the execution.

There is no rule in $\mathcal{GDG}$ permitting a $tailWalker$ or a $minTailWalker$ robot to become another kind of robot. A $headWalker$ can become a $leftWalker$. Let then consider the two following cases.

Case 1: r is a *headWalker* during the whole execution.

Phase `W`**:** A $headWalker$ can execute Rules $\mathbf{T_2}$, $\mathbf{T_3}$ and $\mathbf{W_1}$. Since r does not become a $leftWalker$, it cannot execute Rule $\mathbf{T_2}$. Moreover, since we consider the worst case execution in terms of time, this implies that r is able to execute Rule $\mathbf{W_1}$ entirely. This means that r is able to execute Rule $\mathbf{W_1}$ until its variable $walkSteps$ reaches the value n. In other words, r is able to execute Rule $\mathbf{W_1}$ until it executes Rule $\mathbf{T_3}$.

In this case, the $tailWalker$ and $minTailWalker$ are also able to execute Rule $\mathbf{W_1}$ entirely. Indeed, if, at a time t', while executing Rule $\mathbf{W_1}$ or Rule $\mathbf{T_3}$, the $headWalker$ is waiting on its node for the $tailWalker$ and the $minTailWalker$ to join it while there is an adjacent left edge to its position, and if at time $t'+1$ the $tailWalker$ and the $minTailWalker$ have not join it on its node, this necessarily implies that they stop their execution, otherwise by Rule $\mathbf{W_1}$ they would have join it. Moreover, if such an event happens, r executes Rule $\mathbf{T_1}$ and therefore becomes a $leftWalker$, which leads to a contradiction.

If the $headWalker$ and the $minTailWalker/tailWalker$ execute Rule $\mathbf{W_1}$ entirely, this implies that they move alternatively in the right direction during n steps. Since each edge of the footprint of a $\mathcal{BRE}$ring is present at least once every δ units of time, this takes at most $2 * n * \delta$ rounds. Phase `W` being composed only of the execution of Rule $\mathbf{W_1}$, this phase is bounded.

Phase `T`**:** Call t_v the time at which the $headWalker$ and $minTailWalker/tailWalker$ robots finish to execute Rule $\mathbf{W_1}$ entirely. Since the $headWalker$ and $minTailWalker/tailWalker$ start the execution of Rule $\mathbf{W_1}$ on the same node, at time t_v, they are on the same node v.

Call r_1 and r_2 the two robots that are not involved in the $towerMin$ at time t_{tower}.

If at time t_v, r_1 and r_2 are on node v, then Rule $\mathbf{Term_1}$ is executed at time t_v. In this case, by Lemma 18, Phase `T` last 0 round, hence it is bounded.

If at time t_v, only one robot among r_1 and r_2 is located on node v, then Rule $\mathbf{Term_2}$ is executed at time t_v. Hence, by Lemma 18, $\mathcal{R}-2$ robots are terminated on node v at time t_v. By Lemma 17, at time t_{tower}, r_1 and r_2 are such that $state_{r_1} \in \{righter, potentialMin, awareSearcher, dumbSearcher\}$ and $state_{r_2} \in \{awareSearcher, dumbSearcher\}$. By the movements of the robots given in the proof of Lemma 21, and since each edge of the footprint of a $\mathcal{BRE}$ring is present at least once every δ units of time, it takes at most $n * \delta$ rounds for the last robot to reach node v. Therefore, it takes at most $n * \delta$ rounds for Rule $\mathbf{Term_1}$ to be executed, and thus, by Lemma 18, for all the robots to be terminated on node v. Hence, in this case Phase `T` last at most $n * \delta$ rounds, therefore it is bounded.

Now, consider that at time t_v neither r_1 nor r_2 is located on node v. In this case, at time t_v, the $headWalker$ and $minTailWalker/tailWalker$ execute Rule $\mathbf{T_3}$. While executing Rule $\mathbf{T_3}$, the $headWalker$ (resp. $minTailWalker/tailWalker$) stays a $headWalker$ (resp. $minTailWalker/tailWalker$) and considers the $\perp$ direction. Then, by the rules of $\mathcal{GDG}$, they can only execute Rule $\mathbf{T_3}$ until they terminate. Therefore, they remain on node v from time t_v until the end of their execution. Moreover, as noted previously, by Lemma 17, at time t_{tower}, r_1 and r_2 are such that $state_{r_1} \in \{righter, potentialMin, awareSearcher, dumbSearcher\}$ and $state_{r_2} \in \{awareSearcher, dumbSearcher\}$. By the movements of the robots given in the proof of Lemma 21, since each edge of the footprint of a $\mathcal{BRE}$ ring is present at least once every δ units of time, it takes at most $2 * n * \delta$ rounds for r_1 and r_2 to both reach the node v (in case r_1 and r_2 meet on an adjacent node of v after at most $n * \delta$ rounds of movements in the same direction). In the case the two robots reach node v at the same time, then Rule $\mathbf{Term_1}$ is executed, hence, by Lemma 18, all the robots terminate at that time. In the case the two robots do not reach node v at the same time, then the first one that reaches v permits the execution of Rule $\mathbf{Term_2}$ (hence, by Lemma 18, permits the termination of $\mathcal{R}-2$ robots on node v) and the second that reaches v permits the execution of Rule $\mathbf{Term_1}$. Hence, Phase `T` last at most $2 * n * \delta$ rounds, therefore it is bounded.

Case 2: It exists a time at which r is a *leftWalker*.

By the explanations given in the Case 1, Phase W, at most $2 * n * \delta$ rounds are needed for r to become $leftWalker$ and for the $\mathcal{R} - 2$ other robots to terminate their execution on a node v.

By the rules of $\mathcal{GDG}$, a $leftWalker$ only executes Rule $\mathbf{T_1}$. While executing this rule, a robot considers the $left$ direction and stays a $leftWalker$. Since each edge of the footprint of a $\mathcal{BRE}$ ring is present at least once every δ units of time, such a robot reaches the node v in at most $n * \delta$ rounds. Hence, in this case, Phases W and T take at most $3 * n * \delta$ rounds, hence they are bounded.

Whatever the $\mathcal{BRE}$ ring considered, each phase of $\mathcal{GDG}$ is bounded, therefore, $\mathcal{GDG}$ solves $\mathbb{G}$ in $\mathcal{BRE}$ rings. □

Now, we consider the case of $\mathcal{ST}$ rings. By Lemma 5 and since $\mathcal{ST} \subset \mathcal{BRE}$, we can deduce the following corollary.

Corollary 6. *$\mathcal{GDG}$ solves $\mathbb{G}$ in $\mathcal{ST}$ rings.*

6 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 –refer to Table 1 for a summary of our results. We highlight that it is the first gracefully degrading algorithm dedicated to robot networks and the first algorithm solving –a weak variant of– the gathering problem in $\mathcal{COT}$, the class of dynamic graphs that exhibits the weakest recurrent connectivity.

A natural open question arises on the *optimality* of the gracefully degradation we propose. Indeed, we prove that our algorithm provides for each class of dynamic the best specification *among the ones we proposed*. We do not claim that another algorithm could not be able to satisfy stronger specifications among the infinity of variants one can propose from the original gathering specification. Aside gathering in robots networks, defining and proving formally a general form of *degradation optimality* in the gracefully degrading approach seems to be a challenging future work.

References

[1] Dan Alistarh, Seth Gilbert, Rachid Guerraoui, and Corentin Travers. Generating fast indulgent algorithms. *Theory of Computing Systems (TCS)*, 51(4):404–424, 2012.

[2] Martin Biely, Peter Robinson, Ulrich Schmid, Manfred Schwarz, and Kyrill Winkler. Gracefully degrading consensus and k-set agreement in directed dynamic networks. *Theoretical Computer Science (TCS)*, 726:41–77, 2018.

[3] Marjorie Bournat, Ajoy K. Datta, and Swan Dubois. Self-stabilizing robots in highly dynamic environments. In *18th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 54–69, 2016.

[4] Marjorie Bournat, Swan Dubois, and Franck Petit. Computability of perpetual exploration in highly dynamic rings. In *37th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 794–804, 2017.

[5] Nicolas Braud-Santoni, Swan Dubois, Mohamed-Hamza Kaaouachi, and Franck Petit. The next 700 impossibility results in time-varying graphs. *International Journal of Networking and Computing (IJNC)*, 6(1):27–41, 2016.

[6] Arnaud Casteigts, Paola Flocchini, Bernard Mans, and Nicola Santoro. Shortest, fastest, and foremost broadcast in dynamic networks. *International Journal of Foundations of Computer Science (IJFCS)*, 26(4):499–522, 2015.

[7] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems (IJPEDS)*, 27(5):387–408, 2012.

[8] Swan Dubois and Rachid Guerraoui. Introducing speculation in self-stabilization: an application to mutual exclusion. In *22th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 290–298, 2013.

[9] Partha Dutta and Rachid Guerraoui. The inherent price of indulgence. *Distributed Computing (DC)*, 18(1):85–98, 2005.

[10] Paola Flocchini, Evangelos Kranakis, Danny Krizanc, Nicola Santoro, and Cindy Sawchuk. Multiple mobile agent rendezvous in a ring. In *6th Latin American Symposium on Theoretical Informatics (LATIN)*, volume 2976 of *Lecture Notes in Computer Science*, pages 599–608. Springer, 2004.

[11] Paola Flocchini, Bernard Mans, and Nicola Santoro. On the exploration of time-varying networks. *Theoretical Computer Science (TCS)*, 469:53–68, 2013.

[12] Rachid Guerraoui. Indulgent algorithms (preliminary version). In *9th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 289–297, 2000.

[13] Rachid Guerraoui, Viktor Kuncak, and Giuliano Losa. Speculative linearizability. In *33rd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 55–66, 2012.

[14] David Ilcinkas, Ralf Klasing, and Ahmed Mouhamadou Wade. Exploration of constantly connected dynamic graphs based on cactuses. In *21st International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 250–262, 2014.

[15] Sayaka Kamei, Anissa Lamani, Fukuhito Ooshita, and Sébastien Tixeuil. Asynchronous mobile robot gathering from symmetric configurations without global multiplicity detection. In *18th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 150–161, 2011.

[16] R. Klasing, E. Markou, and A. Pelc. Gathering asynchronous oblivious mobile robots in a ring. In *17th International Symposium on Algorithms and Computation (ISAAC)*, pages 744–753, 2006.

[17] Ramakrishna Kotla, Lorenzo Alvisi, Michael Dahlin, Allen Clement, and Edmund L. Wong. Zyzzyva: Speculative byzantine fault tolerance. *ACM Transactions on Computer Systems (TOCS)*, 27(4):7:1–7:39, 2009.

[18] Fabian Kuhn, Nancy A. Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *42nd ACM Symposium on Theory of Computing (STOC)*, pages 513–522, 2010.

[19] Matthieu Latapy, Tiphaine Viard, and Clémence Magnien. Stream graphs and link streams for the modeling of interactions over time. Technical report, arXiv 1710.04073, 2017.

[20] Giuseppe Antonio Di Luna, Stefan Dobrev, Paola Flocchini, and Nicola Santoro. Live exploration of dynamic rings. In *36th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 570–579, 2016.

[21] Giuseppe Antonio Di Luna, Paola Flocchini, Linda Pagli, Giuseppe Prencipe, Nicola Santoro, and Giovanni Viglietta. Gathering in dynamic rings. In *24th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 339–355, 2017.

[22] Gabriele Di Stefano and Alfredo Navarra. Optimal gathering of oblivious robots in anonymous graphs and its application on trees and rings. *Distributed Computing (DC)*, 30(2):75–86, 2017.

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